

---

# iPod Accessory Protocol Interface Specification

Release R38



2009-10-22



Apple Inc.  
© 2009 Apple Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.  
1 Infinite Loop  
Cupertino, CA 95014  
408-996-1010

iTunes Store is a registered service mark of Apple Inc.

Apple, the Apple logo, FireWire, iPod, iTunes, Mac, Mac OS, Macintosh, and Pages are trademarks of Apple Inc., registered in the United States and other countries.

iPhone, Numbers, QuickStart, and Shuffle are trademarks of Apple Inc.

Times is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

# Contents

## **Introduction      Introduction 29**

---

Organization of This Document 33  
Specification Terms 34  
See Also 34

## **Chapter 1      Protocol Features and Availability 35**

---

General iPod Features 38  
Accessory Power Policy 39  
Video Output Preferences 41  
Character Encoding 42  
Accessory Control of the iPod touch and iPhone 42  
Accessory Communication With iPhone OS Applications 42  
    Setting Up a Communication Session 43  
    Data Flow from the iPod or iPhone 44  
    Matching Accessories With Applications 45  
    Communication Protocol Design Hints 45  
    Communication IAP Commands 46

## **Chapter 2      The Protocol Core and the General Lingo 47**

---

Reserved Commands and Data 47  
Device Signaling and Initialization 48  
    Packet Signaling and Initialization Using the UART Serial Port Link 48  
    iAP Signaling and Initialization Using the USB or BT Port Link 50  
Authentication 52  
    Levels of Device Authentication 53  
    Authentication Requirements 53  
    iPod Authentication of the Accessory 55  
    Device Authentication of iPod 57  
Command Packet Formats 58  
    Small Packet Format 58  
    Large Packet Format 58  
    Packet Details 59  
Lingo 0x00: General Lingo 60  
    General Lingo Command Summary 60  
    History of the General lingo protocol 63  
    Command 0x00: RequestIdentify 64  
    Command 0x02: ACK 65  
    Command 0x03: RequestRemoteUIMode 67  
    Command 0x04: ReturnRemoteUIMode 67

Command 0x05: EnterRemoteUIMode	68
Command 0x06: ExitRemoteUIMode	69
Command 0x07: RequestiPodName	69
Command 0x08: ReturniPodName	70
Command 0x09: RequestiPodSoftwareVersion	70
Command 0x0A: ReturniPodSoftwareVersion	71
Command 0x0B: RequestiPodSerialNum	71
Command 0x0C: ReturniPodSerialNum	72
Command 0x0D: RequestiPodModelNum	72
Command 0x0E: ReturniPodModelNum	73
Command 0x0F: RequestLingoProtocolVersion	79
Command 0x10: ReturnLingoProtocolVersion	80
Command 0x13: IdentifyDeviceLingoes	80
Command 0x14: GetDevAuthenticationInfo	84
Command 0x15: RetDevAuthenticationInfo	84
Command 0x16: AckDevAuthenticationInfo	86
Command 0x17: GetDevAuthenticationSignature	86
Command 0x18: RetDevAuthenticationSignature	87
Command 0x19: AckDevAuthenticationStatus	88
Command 0x1A: GetiPodAuthenticationInfo	89
Command 0x1B: RetiPodAuthenticationInfo	89
Command 0x1C: AckiPodAuthenticationInfo	90
Command 0x1D: GetiPodAuthenticationSignature	90
Command 0x1E: RetiPodAuthenticationSignature	91
Command 0x1F: AckiPodAuthenticationStatus	92
Command 0x23: NotifyiPodStateChange	92
Command 0x24: GetiPodOptions	93
Command 0x25: RetiPodOptions	94
Command 0x27: GetAccessoryInfo	95
Command 0x28: RetAccessoryInfo	97
Command 0x29: GetiPodPreferences	102
Command 0x2A: RetiPodPreferences	106
Command 0x2B: SetiPodPreferences	107
Command 0x38: StartIDPS	108
Command 0x39: SetFIDTokenValues	109
Command 0x3A: RetFIDTokenValueACKs	116
Command 0x3B: EndIDPS	119
Command 0x3C: IDPSStatus	120
Command 0x3F: OpenDataSessionForProtocol	122
Command 0x40: CloseDataSession	123
Command 0x41: DevACK	124
Command 0x42: DevDataTransfer	124
Command 0x43: iPodDataTransfer	126
Command 0x49: SetEventNotification	128
Command 0x4A: iPodNotification	129
Command 0x4B: GetiPodOptionsForLingo	132



Command 0x4C: RetiPodOptionsForLingo	140
Command 0x4D: GetEventNotification	141
Command 0x4E: RetEventNotification	142
Command 0x4F: GetSupportedEventNotification	143
Command 0x51: RetSupportedEventNotification	143

## Chapter 3      **Accessory Lingoes 145**

---

Command Timings	146
Lingo 0x01: Microphone Lingo	147
Command History of the Microphone Lingo	149
Command 0x04: ACK	150
Command 0x05: GetDevAck	151
Command 0x06: iPodModeChange	151
Command 0x07: GetDevCaps	153
Command 0x08: RetDevCaps	153
Command 0x09: GetDevCtrl	154
Command 0x0A: RetDevCtrl	155
Command 0x0B: SetDevCtrl	156
Lingo 0x02: Simple Remote Lingo	157
History and Applicability	157
Playback Engine Playlists	159
Using Contextual Buttons	159
Using Dedicated Media Buttons	160
Accessory Control of the iPod 5G nano Camera	161
Command 0x00: ContextButtonStatus	167
Command 0x01: ACK	169
Command 0x02: ImageButtonStatus	170
Command 0x03: VideoButtonStatus	171
Command 0x04: AudioButtonStatus	173
Command 0x0D: RadioButtonStatus	174
Command 0x0E: CameraButtonStatus	175
Lingo 0x03: Display Remote Lingo	176
Command History of the Display Remote lingo	178
Transferring Album Art	179
Command 0x00: ACK	180
Command 0x01: GetCurrentEQProfileIndex	181
Command 0x02: RetCurrentEQProfileIndex	181
Command 0x03: SetCurrentEQProfileIndex	182
Command 0x04: GetNumEQProfiles	183
Command 0x05: RetNumEQProfiles	183
Command 0x06: GetIndexedEQProfileName	184
Command 0x07: RetIndexedEQProfileName	184
Command 0x08: SetRemoteEventNotification	185
Command 0x09: RemoteEventNotification	187
Command 0x0A: GetRemoteEventStatus	193

Command 0x0B: RetRemoteEventStatus	194
Command 0x0C: GetiPodStateInfo	194
Command 0x0D: RetiPodStateInfo	196
Command 0x0E: SetiPodStateInfo	197
Command 0x0F: GetPlayStatus	203
Command 0x10: RetPlayStatus	203
Command 0x11: SetCurrentPlayingTrack	204
Command 0x12: GetIndexedPlayingTrackInfo	205
Command 0x13: RetIndexedPlayingTrackInfo	206
Command 0x14: GetNumPlayingTracks	208
Command 0x15: RetNumPlayingTracks	209
Command 0x16: GetArtworkFormats	209
Command 0x17: RetArtworkFormats	210
Command 0x18: GetTrackArtworkData	211
Command 0x19: RetTrackArtworkData	212
Command 0x1A: GetPowerBatteryState	213
Command 0x1B: RetPowerBatteryState	214
Command 0x1C: GetSoundCheckState	214
Command 0x1D: RetSoundCheckState	215
Command 0x1E: SetSoundCheckState	215
Command 0x1F: GetTrackArtworkTimes	216
Command 0x20: RetTrackArtworkTimes	217
Lingo 0x04: Extended Interface Lingo	218
Lingo 0x05: Accessory Power Lingo	218
Command History of the Accessory Power lingo	219
Command 0x02: BeginHighPower	219
Command 0x03: EndHighPower	219
Lingo 0x07: RF Tuner Lingo	220
RF Tuner Accessory Design	220
RF Tuner Power	221
RF Tuner Lingo Commands	221
Command History of the RF Tuner Lingo	223
Command 0x00: ACK	224
Command 0x01: GetTunerCaps	226
Command 0x02: RetTunerCaps	226
Command 0x03: GetTunerCtrl	228
Command 0x04: RetTunerCtrl	229
Command 0x05: SetTunerCtrl	229
Command 0x06: GetTunerBand	231
Command 0x07: RetTunerBand	231
Command 0x08: SetTunerBand	232
Command 0x09: GetTunerFreq	232
Command 0x0A: RetTunerFreq	233
Command 0x0B: SetTunerFreq	234
Command 0x0C: GetTunerMode	234
Command 0x0D: RetTunerMode	235

Command 0x0E: SetTunerMode	236
Command 0x0F: GetTunerSeekRssi	237
Command 0x10: RetTunerSeekRssi	237
Command 0x11: SetTunerSeekRssi	238
Command 0x12: TunerSeekStart	238
Command 0x13: TunerSeekDone	240
Command 0x14: GetTunerStatus	241
Command 0x15: RetTunerStatus	242
Command 0x16: GetStatusNotifyMask	243
Command 0x17: RetStatusNotifyMask	243
Command 0x18: SetStatusNotifyMask	244
Command 0x19: StatusChangeNotify	245
Command 0x1A: GetRdsReadyStatus	247
Command 0x1B: RetRdsReadyStatus	247
Command 0x1C: GetRdsData	248
Command 0x1D: RetRdsData	249
Command 0x1E: GetRdsNotifyMask	251
Command 0x1F: RetRdsNotifyMask	252
Command 0x20: SetRdsNotifyMask	253
Command 0x21: RdsReadyNotify	254
Command 0x25: GetHDPProgramServiceCount	255
Command 0x26: RetHDPProgramServiceCount	255
Command 0x27: GetHDPProgramService	256
Command 0x28: RetHDPProgramService	256
Command 0x29: SetHDPProgramService	257
Command 0x2A: GetHDDDataReadyStatus	258
Command 0x2B: RetHDDDataReadyStatus	258
Command 0x2C: GetHDDData	259
Command 0x2D: RetHDDData	260
Command 0x2E: GetHDDDataNotifyMask	262
Command 0x2F: RetHDDDataNotifyMask	263
Command 0x30: SetHDDDataNotifyMask	264
Command 0x31: HDDDataReadyNotify	265
Sample HD Command Sequences	267
Lingo 0x08: Accessory Equalizer Lingo	271
Equalizer Setting Requirements	271
Accessory Equalizer Lingo Commands	271
Command 0x00: ACK	272
Command 0x01: GetCurrentEQIndex	273
Command 0x02: RetCurrentEQIndex	273
Command 0x03: SetCurrentEQIndex	274
Command 0x04: GetEQSettingCount	274
Command 0x05: RetEQSettingCount	275
Command 0x06: GetEQIndexName	276
Command 0x07: RetEQIndexName	276
Lingo 0x09: Sports Lingo	277

Sports Lingo commands	277
Command History of the Sports Lingo	278
Command 0x00: DeviceACK	278
Command 0x01: GetDeviceVersion	279
Command 0x02: RetDeviceVersion	280
Command 0x03: GetDeviceCaps	280
Command 0x04: RetDeviceCaps	281
Command 0x80: iPodACK	282
Command 0x83: GetiPodCaps	283
Command 0x84: RetiPodCaps	283
Command 0x85: GetUserIndex	284
Command 0x86: RetUserIndex	285
Command 0x88: GetUserData	285
Command 0x89: RetUserData	286
Command 0x8A: SetUserData	288
Lingo 0x0A: Digital Audio Lingo	289
Accessory Authentication	290
USB Audio Transport	291
Command History of the Digital Audio Lingo	294
Command 0x00: AccAck	295
Command 0x01: iPodAck	296
Command 0x02: GetAccSampleRateCaps	296
Command 0x03: RetAccSampleRateCaps	297
Command 0x04: NewiPodTrackInfo	298
Command 0x05: SetVideoDelay	299
Lingo 0x0C: Storage Lingo	300
Command History of the Storage Lingo	300
Command Summary	300
Command 0x00: iPodACK	302
Command 0x01: GetiPodCaps	303
Command 0x02: RetiPodCaps	303
Command 0x04: RetiPodFileHandle	305
Command 0x07: WriteiPodFileData	305
Command 0x08: CloseiPodFile	306
Command 0x10: GetiPodFreeSpace	307
Command 0x11: RetiPodFreeSpace	307
Command 0x12: OpeniPodFeatureFile	308
Command 0x80: DeviceACK	310
Command 0x81: GetDeviceCaps	311
Command 0x82: RetDeviceCaps	311
Lingo 0x0E: Location Lingo	312
Location Data Requirements	312
Accessory Power Using the Location Lingo	313
Command Summary	313
A Typical Location Data Session	315
Command 0x00: DevACK	317

Command 0x01: GetDevCaps	318
Command 0x02: RetDevCaps	319
Command 0x03: GetDevControl	322
Command 0x04: RetDevControl	322
Command 0x05: SetDevControl	323
Command 0x06: GetDevData	325
Command 0x07: RetDevData	326
Command 0x08: SetDevData	328
Command 0x09: AsyncDevData	331
Command 0x80: iPodACK	333
Sample Identification Sequences	334

## Chapter 4      **The Extended Interface Protocol   341**

---

Major iPod Operating Modes	341
Extended Interface Mode	342
Sleep State	342
Packet Formats	343
Small Packet Format	343
Large Packet Format	343
Packet Details	344
Accessory Identification and Authentication	344
Command Transports	347
Entering Extended Interface Mode	347
Using the Extended Interface Protocol	347
The Playback Engine	348
The Database Engine	350
Transferring Album Art	354
Using the Extended Interface Protocol: An Example	355
Video Browsing	358
Sample Extended Interface Session	364
Command Packets	367
Command Code Summary	367
Command 0x0001: ACK	371
Command 0x0002: GetCurrentPlayingTrackChapterInfo	372
Command 0x0003: ReturnCurrentPlayingTrackChapterInfo	373
Command 0x0004: SetCurrentPlayingTrackChapter	374
Command 0x0005: GetCurrentPlayingTrackChapterPlayStatus	375
Command 0x0006: ReturnCurrentPlayingTrackChapterPlayStatus	375
Command 0x0007: GetCurrentPlayingTrackChapterName	376
Command 0x0008: ReturnCurrentPlayingTrackChapterName	377
Command 0x0009: GetAudiobookSpeed	378
Command 0x000A: ReturnAudiobookSpeed	378
Command 0x000B: SetAudiobookSpeed	379
Command 0x000C: GetIndexedPlayingTrackInfo	380
Command 0x000D: ReturnIndexedPlayingTrackInfo	382

Command 0x000E: GetArtworkFormats	385
Command 0x000F: RetArtworkFormats	386
Command 0x0010: GetTrackArtworkData	387
Command 0x0011: RetTrackArtworkData	387
Command 0x0016: ResetDBSelection	389
Command 0x0017: SelectDBRecord	390
Command 0x0018: GetNumberCategorizedDBRecords	392
Command 0x0019: ReturnNumberCategorizedDBRecords	393
Command 0x001A: RetrieveCategorizedDatabaseRecords	393
Command 0x001B: ReturnCategorizedDatabaseRecord	394
Command 0x001C: GetPlayStatus	395
Command 0x001D: ReturnPlayStatus	396
Command 0x001E: GetCurrentPlayingTrackIndex	397
Command 0x001F: ReturnCurrentPlayingTrackIndex	398
Command 0x0020: GetIndexedPlayingTrackTitle	398
Command 0x0021: ReturnIndexedPlayingTrackTitle	399
Command 0x0022: GetIndexedPlayingTrackArtistName	400
Command 0x0023: ReturnIndexedPlayingTrackArtistName	401
Command 0x0024: GetIndexedPlayingTrackAlbumName	401
Command 0x0025: ReturnIndexedPlayingTrackAlbumName	402
Command 0x0026: SetPlayStatusChangeNotification	403
Command 0x0027: PlayStatusChangeNotification	405
Command 0x0028: PlayCurrentSelection	407
Command 0x0029: PlayControl	407
Command: 0x002A: GetTrackArtworkTimes	409
Command: 0x002B: RetTrackArtworkTimes	410
Command 0x002C: GetShuffle	411
Command 0x002D: ReturnShuffle	412
Command 0x002E: SetShuffle	412
Command 0x002F: GetRepeat	414
Command 0x0030: ReturnRepeat	414
Command 0x0031: SetRepeat	415
Command 0x0032: SetDisplayImage	416
Command 0x0033: GetMonoDisplayImageLimits	422
Command 0x0034: ReturnMonoDisplayImageLimits	422
Command 0x0035: GetNumPlayingTracks	423
Command 0x0036: ReturnNumPlayingTracks	424
Command 0x0037: SetCurrentPlayingTrack	424
Command 0x0038: SelectSortDBRecord	425
Command 0x0039: GetColorDisplayImageLimits	427
Command 0x003A: ReturnColorDisplayImageLimits	428
Command 0x003B: ResetDBSelectionHierarchy	429
Command 0x003C: GetDBiTunesInfo	430
Command 0x003D: RetDBiTunesInfo	431
Command 0x003E: GetUIDTrackInfo	432
Command 0x003F: RetUIDTrackInfo	434

Command 0x0040: GetDBTrackInfo	437
Command 0x0041: RetDBTrackInfo	438
Command 0x0042: GetPBTrackInfo	439
Command 0x0043: RetPBTrackInfo	440
Known Issues	441
Protocol History	441

## Appendix A      **Accessory Identification 445**

---

Using IDPS	445
IDPS Commands	446
Sample IDPS Command Sequences	446
iPod Event Notifications	449

## Appendix B      **Transaction IDs 451**

---

Using Transaction IDs	451
Generating and Returning Transaction IDs	451
Enabling and Disabling Transaction ID Support	451
iPod Acknowledgment of Transaction ID Commands	452
Command 0x02: General Lingo ACK	452
Examples of Transaction IDs	456

## Appendix C      **Multisection Data Transfers 459**

---

Multisection Transfer Process	459
Multisection iPodACK Command	460
Multisection DevACK Command	460

## Appendix D      **iTunes Tagging 463**

---

The iTunes Tagging Experience	463
Tagging Feature Components	463
Radio Accessory Requirements	465
HD Radio Tagging	465
FM Radio Tagging	466
Handling Unknown Metadata Types	475
Resolving Tag Ambiguity	476
Capturing and Storing Tag Data	477
Tag Data Writing Process	477
Data Transfer to the iPod	479
Accessory User Interface	482
Tag Button Text	484
Sample Command Sequence	484
Sample Tag Files	488
HD Radio Samples	488

FM Radio Sample 492

## Appendix E **Nike + iPod Cardio Equipment System 495**

---

- Using Nike + iPod Cardio Equipment 495
- Cardio Equipment Design 495
  - Determining iPod Support for Cardio Equipment 495
  - Identification Procedure 496
  - Declaring Cardio Equipment Support to the iPod 498
  - Accessing User Data 499
  - Recording Workout Data 499
  - XML Element Formats 502
- User Interface Requirements 506
  - The iPod Does Not Support Nike + iPod 507
  - Deciding to Record a Workout to the iPod 507
  - The iPod is Full 507
  - User Weight 507
  - Recording Indicator 508
  - Workout Completed 508
- Sample Command Sequence 510

## Appendix F **Historical Information 517**

---

- Past Protocol Features 517
- 9-Pin Audio/Remote Connector Commands 520
  - Command 0x00: BeginRecord 521
  - Command 0x01: EndRecord 521
  - Command 0x02: BeginPlayback 522
  - Command 0x03: EndPlayback 522
- General Lingo Command 0x01: Identify (Deprecated) 523
- Lingo 0x06: USB Host Control Lingo 525
  - Command History of the USB Host Control Lingo 525
  - Command 0x00: ACK 526
  - Command 0x01: GetUSBPowerState 526
  - Command 0x02: RetUSBPowerState 527
  - Command 0x03: SetUSBPowerState 527
- Deprecated Extended Interface Commands 528
  - Command 0x0012: RequestProtocolVersion 528
  - Command 0x0013: ReturnProtocolVersion 528
  - Command 0x0014: RequestiPodName 529
  - Command 0x0015: ReturniPodName 530
- Authentication 1.0 Sample Command Sequence 531
- Accessory Identification With Non-IDPS iPods 532



[Glossary 555](#)

---

[Document Revision History 557](#)

---



# Figures, Tables, and Listings

## Introduction      **Introduction 29**

---

Table I-1      iPod and iPhone products 30

## Chapter 1      **Protocol Features and Availability 35**

---

Table 1-1      Current firmware and OS versions in iPod and iPhone models 35  
Table 1-2      Most recent iPod models, firmware, and lingo versions, Table 1 36  
Table 1-3      Most recent iPod/iPhone models, firmware, and lingo versions, Table 2 37  
Table 1-4      Features supported by specific iPod firmware and OS versions, Table 1 38  
Table 1-5      Features supported by specific iPod/iPhone firmware and OS versions, Table 2 39  
Table 1-6      Video output capabilities 41  
Table 1-7      Accessory communication commands 46

## Chapter 2      **The Protocol Core and the General Lingo 47**

---

Figure 2-1      Screen configuration examples 106  
Table 2-1      Command traffic for UART accessory identification 48  
Table 2-2      Commands for accessory identification via USB or BT 51  
Table 2-3      iPod and iPhone authentication coprocessor classes 53  
Table 2-4      Lingo commands requiring authentication 54  
Table 2-5      Command traffic for accessory authentication 55  
Table 2-6      Accessory authentication of the iPod 57  
Table 2-7      Small packet format 58  
Table 2-8      Large packet format 59  
Table 2-9      General lingo commands 60  
Table 2-10      General lingo revision history 64  
Table 2-11      RequestIdentify packet 64  
Table 2-12      ACK packet 65  
Table 2-13      ACK command error codes 65  
Table 2-14      ACK packet with Command Pending status 66  
Table 2-15      RequestRemoteUIMode packet 67  
Table 2-16      ReturnRemoteUIMode packet 67  
Table 2-17      EnterRemoteUIMode packet 68  
Table 2-18      ExitRemoteUIMode packet 69  
Table 2-19      RequestiPodName packet 69  
Table 2-20      ReturniPodName packet 70  
Table 2-21      RequestiPodSoftwareVersion packet 70  
Table 2-22      ReturniPodSoftwareVersion packet 71  
Table 2-23      RequestiPodSerialNum packet 71  
Table 2-24      ReturniPodSerialNum packet 72  
Table 2-25      RequestiPodModelNum packet 72

Table 2-26	ReturniPodModelNum packet	73
Table 2-27	iPod model IDs	74
Table 2-28	iPod model number strings M8948-M9974 and P8948-P9830	75
Table 2-29	iPod model number strings MA002/PA002 and higher	76
Table 2-30	RequestLingoProtocolVersion packet	80
Table 2-31	ReturnLingoProtocolVersion packet	80
Table 2-32	IdentifyDeviceLingoes packet	81
Table 2-33	Device Lingoes Spoken bits	82
Table 2-34	IdentifyDeviceLingoes Options bits	83
Table 2-35	GetDevAuthenticationInfo packet	84
Table 2-36	RetDevAuthenticationInfo packet, Authentication 1.0	85
Table 2-37	RetDevAuthenticationInfo packet, Authentication 2.0	85
Table 2-38	AckDevAuthenticationInfo packet	86
Table 2-39	GetDevAuthenticationSignature packet	87
Table 2-40	RetDevAuthenticationSignature packet	88
Table 2-41	AckDevAuthenticationStatus packet	88
Table 2-42	GetiPodAuthenticationInfo packet	89
Table 2-43	RetiPodAuthenticationInfo packet	89
Table 2-44	AckiPodAuthenticationInfo packet	90
Table 2-45	GetiPodAuthenticationSignature packet	91
Table 2-46	RetiPodAuthenticationSignature packet	91
Table 2-47	AckiPodAuthenticationStatus packet	92
Table 2-48	NotifyiPodStateChange packet	93
Table 2-49	GetiPodOptions packet	93
Table 2-50	RetiPodOptions packet	94
Table 2-51	GetAccessoryInfo packet	95
Table 2-52	Accessory Info Type values	96
Table 2-53	GetAccessoryInfo packet with Accessory Info Type = 0x02	96
Table 2-54	GetAccessoryInfo packet with Accessory Info Type = 0x03	97
Table 2-55	RetAccessoryInfo packet with Accessory Info Type = 0x00	97
Table 2-56	Accessory Info Type values for RetAccessoryInfo	98
Table 2-57	Accessory Capabilities bit field	98
Table 2-58	RetAccessoryInfo packet with Accessory Info Type = 0x01/0x06/0x07/0x08	99
Table 2-59	RetAccessoryInfo packet with Accessory Info Type = 0x02	100
Table 2-60	RetAccessoryInfo packet with Accessory Info Type = 0x03	100
Table 2-61	RetAccessoryInfo packet with Accessory Info Type = 0x04/0x05	101
Table 2-62	RetAccessoryInfo packet with Accessory Info Type = 0x09	101
Table 2-63	GetiPodPreferences packet	102
Table 2-64	iPod preference class and setting IDs	103
Table 2-65	RetiPodPreferences packet	107
Table 2-66	SetiPodPreferences packet	108
Table 2-67	StartIDPS packet	109
Table 2-68	SetFIDTokenValues packet	110
Table 2-69	FIDTokenValues field format	110
Table 2-70	FIDTokenValues tokens	111
Table 2-71	IdentifyToken format	112

Table 2-72	DeviceOptions bits in IdentifyToken	112
Table 2-73	AccCapsToken format	113
Table 2-74	Accessory capabilities bit values	113
Table 2-75	AccInfoToken format	113
Table 2-76	Accessory Info Type values	114
Table 2-77	iPodPreferenceToken format	114
Table 2-78	SDKProtocolToken format	114
Table 2-79	BundleSeedIDPrefToken format	115
Table 2-80	MicrophoneCapsToken format	115
Table 2-81	Microphone capabilities bits	115
Table 2-82	RetFIDTokenValueACKs packet	116
Table 2-83	Acknowledgment format for IdentifyToken	117
Table 2-84	Acknowledgment status codes	117
Table 2-85	Acknowledgment format for AccCapsToken	117
Table 2-86	Acknowledgment format for AccInfoToken	118
Table 2-87	Acknowledgment format for iPodPreferenceToken	118
Table 2-88	Acknowledgment format for SDKProtocolToken	118
Table 2-89	Acknowledgment format for BundleSeedIDPrefToken	118
Table 2-90	Acknowledgment format for MicCapsToken	119
Table 2-91	EndIDPS packet	119
Table 2-92	accEndIDPSStatus values	119
Table 2-93	IDPSStatus packet	120
Table 2-94	iPod actions in response to accEndIDPSStatus and IDPS status	121
Table 2-95	OpenDataSessionForProtocol packet	122
Table 2-96	CloseDataSession packet	123
Table 2-97	DevACK packet	124
Table 2-98	DevDataTransfer small packet	125
Table 2-99	DevDataTransfer large packet	125
Table 2-100	iPodDataTransfer small packet	127
Table 2-101	iPodDataTransfer large packet	127
Table 2-102	SetEventNotification packet	128
Table 2-103	Notification bitmask bits	129
Table 2-104	iPodNotification packet for Flow Control Notifications	130
Table 2-105	iPodNotification packet for Radio Tagging Notifications	131
Table 2-106	iPodNotification payload for Radio Tagging notifications	131
Table 2-107	iPodNotification packet for Camera notifications	132
Table 2-108	iPodNotification payload for Camera Notifications	132
Table 2-109	GetiPodOptionsForLingo packet	133
Table 2-110	RetiPodOptionsForLingo option bits	133
Table 2-111	Sample GetiPodOptionsForLingo and RetiPodOptionsForLingo commands	135
Table 2-112	RetiPodOptionsForLingo packet	141
Table 2-113	GetEventNotification packet	141
Table 2-114	RetEventNotification packet	142
Table 2-115	GetSupportedEventNotification packet	143
Table 2-116	RetSupportedEventNotification packet	143

## Chapter 3

## Accessory Lingo 145

---

Figure 3-1	A USB host recovers from a CRC16 error	294
Table 3-1	iPod accessory lingo	145
Table 3-2	Select iPod command timings	146
Table 3-3	Microphone lingo command summary	149
Table 3-4	Microphone lingo command history	149
Table 3-5	ACK packet	150
Table 3-6	Command result values	150
Table 3-7	GetDevAck packet	151
Table 3-8	iPodModeChange packet	152
Table 3-9	Mode values	152
Table 3-10	GetDevCaps packet	153
Table 3-11	RetDevCaps packet	154
Table 3-12	Microphone capabilities bitmask	154
Table 3-13	GetDevCtrl packet	155
Table 3-14	RetDevCtrl packet	155
Table 3-15	Control types and data	156
Table 3-16	SetDevCtrl packet	157
Table 3-17	Simple remote lingo command summary	157
Table 3-18	Simple remote lingo support versions	158
Table 3-19	Simple Remote lingo command history	158
Table 3-20	iPod camera modes and transitions	161
Table 3-21	Suggested accessory feedback indications	162
Table 3-22	iPod camera interface commands	163
Table 3-23	Sample video control commands, accessory on and off	164
Table 3-24	Sample video control commands, iPod on and accessory off	166
Table 3-25	ContextButtonStatus packet	167
Table 3-26	Button states	168
Table 3-27	ACK packet	169
Table 3-28	Command status codes	169
Table 3-29	ImageButtonStatus packet	170
Table 3-30	Image-specific button values	171
Table 3-31	VideoButtonStatus packet	171
Table 3-32	Video-specific button values	172
Table 3-33	AudioButtonStatus packet	173
Table 3-34	Audio-specific button values	173
Table 3-35	RadioButtonStatus packet	175
Table 3-36	CameraButtonStatus packet	176
Table 3-37	Display Remote lingo command summary	177
Table 3-38	Display Remote lingo command history	178
Table 3-39	ACK packet	180
Table 3-40	Command result values	180
Table 3-41	GetCurrentEQProfileIndex packet	181
Table 3-42	RetCurrentEQProfileIndex packet	181
Table 3-43	SetCurrentEQProfileIndex packet	182

Table 3-44	GetNumEQProfiles packet	183
Table 3-45	RetNumEQProfiles packet	183
Table 3-46	GetIndexedEQProfileName packet	184
Table 3-47	RetIndexedEQProfileName packet	185
Table 3-48	SetRemoteEventNotification packet	185
Table 3-49	iPod events	186
Table 3-50	RemoteEventNotification packet	187
Table 3-51	Event notification data	188
Table 3-52	Play status values	191
Table 3-53	Shuffle state	192
Table 3-54	Repeat state	192
Table 3-55	Power and battery state	192
Table 3-56	Audiobook playback speeds	193
Table 3-57	GetRemoteEventStatus packet	193
Table 3-58	RetRemoteEventStatus packet	194
Table 3-59	GetiPodStateInfo packet	195
Table 3-60	infoType values	195
Table 3-61	GetiPodStateInfo packet to retrieve the iPod Equalizer Setting	196
Table 3-62	RetiPodStateInfo packet	196
Table 3-63	RetiPodStateInfo packet for requesting chapter information	197
Table 3-64	SetiPodStateInfo packet	198
Table 3-65	iPod state data	198
Table 3-66	Restore-on-exit values	202
Table 3-67	SetiPodStateInfo packet to set the alarm	202
Table 3-68	SetiPodStateInfo packet for setting the current track	202
Table 3-69	GetPlayStatus packet	203
Table 3-70	RetPlayStatus packet	204
Table 3-71	SetCurrentPlayingTrack packet	205
Table 3-72	GetIndexedPlayingTrackInfo packet	205
Table 3-73	RetIndexedPlayingTrackInfo packet	206
Table 3-74	Track information data	207
Table 3-75	GetNumPlayingTracks packet	209
Table 3-76	RetNumPlayingTracks packet	209
Table 3-77	GetArtworkFormats packet	210
Table 3-78	RetArtworkFormats packet	210
Table 3-79	Display pixel format codes	211
Table 3-80	GetTrackArtworkData packet	211
Table 3-81	RetTrackArtworkData packet	212
Table 3-82	GetPowerBatteryState packet	213
Table 3-83	RetPowerBatteryState packet	214
Table 3-84	GetSoundCheckState packet	215
Table 3-85	RetSoundCheckState packet	215
Table 3-86	SetSoundCheckState packet	216
Table 3-87	GetTrackArtworkTimes packet	216
Table 3-88	RetTrackArtworkTimes packet	217
Table 3-89	Accessory Power lingo command summary	218

Table 3-90	Accessory Power lingo command history	219
Table 3-91	BeginHighPower packet	219
Table 3-92	EndHighPower packet	220
Table 3-93	RF Tuner lingo command summary	222
Table 3-94	RF Tuner lingo command history	224
Table 3-95	RF tuner lingo ACK packet with cmdStatus not 0x06	224
Table 3-96	RF tuner lingo ACK packet with cmdStatus equal to 0x06	224
Table 3-97	RF tuner lingo ACK command status values	225
Table 3-98	GetTunerCaps packet	226
Table 3-99	RetTunerCaps packet	226
Table 3-100	RF tuner device capabilities payload	227
Table 3-101	Minimum FM resolution ID bits	228
Table 3-102	GetTunerCtrl packet	228
Table 3-103	RetTunerCtrl packet	229
Table 3-104	Tuner control state bits	229
Table 3-105	SetTunerCtrl packet	230
Table 3-106	Tuner control bits	230
Table 3-107	GetTunerBand packet	231
Table 3-108	RetTunerBand packet	231
Table 3-109	Tuner band state IDs	232
Table 3-110	SetTunerBand packet	232
Table 3-111	GetTunerFreq packet	233
Table 3-112	RetTunerFreq packet	233
Table 3-113	SetTunerFreq packet	234
Table 3-114	GetTunerMode packet	235
Table 3-115	RetTunerMode packet	235
Table 3-116	RF Tuner mode status bits	235
Table 3-117	SetTunerMode packet	236
Table 3-118	Set RF Tuner mode bits	236
Table 3-119	GetTunerSeekRssi packet	237
Table 3-120	RetTunerSeekRssi packet	237
Table 3-121	SetTunerSeekRssi packet	238
Table 3-122	TunerSeekStart packet	239
Table 3-123	Tuner seeking operations	239
Table 3-124	TunerSeekDone packet	241
Table 3-125	GetTunerStatus packet	241
Table 3-126	RetTunerStatus packet	242
Table 3-127	RF tuner status bits	242
Table 3-128	GetStatusNotifyMask packet	243
Table 3-129	RetStatusNotifyMask packet	243
Table 3-130	Status notification mask bits	244
Table 3-131	SetStatusNotifyMask packet	245
Table 3-132	Status notification mask setting bits	245
Table 3-133	StatusChangeNotify packet	246
Table 3-134	Status change ID bits	246
Table 3-135	GetRdsReadyStatus packet	247



Table 3-136	RetRdsReadyStatus packet 247
Table 3-137	RDS/RBDS data-ready status bits, parsed mode 248
Table 3-138	RDS/RBDS data-ready status bits, raw mode 248
Table 3-139	GetRdsData packet 248
Table 3-140	RDS/RBDS data type IDs, parsed mode 249
Table 3-141	RDS/RBDS data type IDs, raw mode 249
Table 3-142	RetRdsData packet 249
Table 3-143	rdsDataType bytes and rdsData formats 250
Table 3-144	rdsData character set IDs 250
Table 3-145	RDS/RBDS group data-ready data 251
Table 3-146	Block errors byte encoding 251
Table 3-147	Block error bit values 251
Table 3-148	GetRdsNotifyMask packet 252
Table 3-149	RetRdsNotifyMask packet 252
Table 3-150	RDS/RBDS data change notification mask bits, parsed mode 253
Table 3-151	RDS/RBDS data change notification mask bits, raw mode 253
Table 3-152	SetRdsNotifyMask packet 253
Table 3-153	RDS/RBDS data change notification mask setting bits, parsed mode 254
Table 3-154	RDS/RBDS data change notification mask setting bits, raw mode 254
Table 3-155	RdsReadyNotify packet 254
Table 3-156	GetHDPProgramServiceCount packet 255
Table 3-157	RetHDPProgramServiceCount packet 256
Table 3-158	GetHDPProgramService packet 256
Table 3-159	RetHDPProgramService packet 257
Table 3-160	SetHDPProgramService packet 257
Table 3-161	GetHDDDataReadyStatus packet 258
Table 3-162	RetHDDDataReadyStatus packet 258
Table 3-163	HD data-ready status bits 259
Table 3-164	GetHDDData packet 260
Table 3-165	HD data type IDs 260
Table 3-166	RetHDDData packet 261
Table 3-167	HDDDataType type IDs and formats 261
Table 3-168	PSD data format 262
Table 3-169	8-bit character encoding type formats 262
Table 3-170	GetHDDDataNotifyMask packet 262
Table 3-171	RetHDDDataNotifyMask packet 263
Table 3-172	HD data change notification mask bits 263
Table 3-173	SetHDDDataNotifyMask packet 264
Table 3-174	HD data change notification mask setting bits 264
Table 3-175	HDDDataReadyNotify packet 265
Table 3-176	HD data types 265
Table 3-177	HD Data Type type IDs and formats 266
Table 3-178	PSD data format 266
Table 3-179	8-bit character encoding type formats 267
Table 3-180	Example of HD radio setup 267
Table 3-181	Example of HD radio tuning and reception 268

Table 3-182	Example of getting PSD and name data	270
Table 3-183	Accessory Equalizer lingo command summary	271
Table 3-184	Accessory Equalizer lingo command history	272
Table 3-185	ACK packet	272
Table 3-186	GetCurrentEQIndex packet	273
Table 3-187	RetCurrentEQIndex packet	273
Table 3-188	Device Equalizer Setting indices	274
Table 3-189	SetCurrentEQIndex packet	274
Table 3-190	GetEQSettingCount packet	275
Table 3-191	RetEQSettingCount packet	275
Table 3-192	RetEQSettingCount parameter values	275
Table 3-193	GetEQIndexName packet	276
Table 3-194	RetEQIndexName packet	276
Table 3-195	Sports lingo command summary	277
Table 3-196	Sports lingo command history	278
Table 3-197	DeviceACK packet	278
Table 3-198	ackStatus details	279
Table 3-199	GetDeviceVersion packet	279
Table 3-200	RetDeviceVersion packet	280
Table 3-201	GetDeviceCaps packet	281
Table 3-202	RetDeviceCaps packet	281
Table 3-203	RetDeviceCaps capsMask details	282
Table 3-204	iPodACK packet	282
Table 3-205	iPodACK ackStatus details	282
Table 3-206	GetiPodCaps packet	283
Table 3-207	RetiPodCaps packet	283
Table 3-208	RetiPodCaps capsMask details	284
Table 3-209	GetUserIndex packet	284
Table 3-210	RetUserIndex packet	285
Table 3-211	GetUserData packet	286
Table 3-212	GetUserData userDataType details	286
Table 3-213	RetUserData packet	287
Table 3-214	RetUserData userDataType details	287
Table 3-215	SetUserData packet	288
Table 3-216	SetUserData userDataType details	289
Table 3-217	Digital Audio lingo command summary	290
Table 3-218	Digital Audio lingo command history	294
Table 3-219	AccAck packet	295
Table 3-220	AccAck status values	295
Table 3-221	iPodAck packet	296
Table 3-222	GetAccSampleRateCaps packet	296
Table 3-223	RetAccSampleRateCaps packet	297
Table 3-224	Digital audio sample rates supported by iPods (in Hertz)	298
Table 3-225	NewiPodTrackInfo packet	299
Table 3-226	SetVideoDelay packet	300
Table 3-227	Storage lingo command history	300

Table 3-228	Storage lingo commands	301
Table 3-229	Storage iPodACK packet	302
Table 3-230	iPodACK responses	302
Table 3-231	GetiPodCaps packet	303
Table 3-232	RetiPodCaps packet	304
Table 3-233	RetiPodFileHandle packet	305
Table 3-234	WriteiPodFileData packet	306
Table 3-235	CloseiPodFile packet	307
Table 3-236	GetiPodFreeSpace packet	307
Table 3-237	RetiPodFreeSpace packet	308
Table 3-238	OpeniPodFeatureFile packet	308
Table 3-239	featureType values	309
Table 3-240	fileOptionsMask values	309
Table 3-241	DeviceACK packet	310
Table 3-242	ackStatus values	310
Table 3-243	GetDeviceCaps packet	311
Table 3-244	RetDeviceCaps packet	311
Table 3-245	Location lingo commands	314
Table 3-246	Sample command interchange	315
Table 3-247	Default DevACK packet	317
Table 3-248	Valid Location lingo ackStatus values	318
Table 3-249	GetDevCaps packet	318
Table 3-250	Values for locType	319
Table 3-251	RetDevCaps packet	319
Table 3-252	System capabilities values (locType = 0x00)	320
Table 3-253	NMEA GPS location capabilities values (locType = 0x01)	321
Table 3-254	Location assistance capabilities values (locType = 0x02)	321
Table 3-255	GetDevControl packet	322
Table 3-256	RetDevControl packet	322
Table 3-257	SetDevControl packet	323
Table 3-258	ctlData values	324
Table 3-259	GetDevData packet	325
Table 3-260	GetDevData dataType values	325
Table 3-261	RetDevData packet	326
Table 3-262	RetDevData locData values	327
Table 3-263	locAsstData values (locType = 0x02) for RetDevData	327
Table 3-264	SetDevData packet	328
Table 3-265	Data types settable by SetDevData	329
Table 3-266	nmeaGpsLocData values (locType = 0x01)	329
Table 3-267	locAsstData values (locType = 0x02) for SetDevData	330
Table 3-268	AsyncDevData packet	332
Table 3-269	locData values that can be sent by AsyncDevData	333
Table 3-270	Default iPodACK packet	333
Table 3-271	Identification of lingoes 0x00+0x04	334
Table 3-272	Identification of lingoes 0x00+0x02+0x03	335
Table 3-273	Identification of lingoes 0x00+0x02+0x03+0x0A	337

## Chapter 4      The Extended Interface Protocol   341

---

Figure 4-1	Typical “OK to disconnect” screens   342
Figure 4-2	An example interaction between an Extended Interface device and an iPod   356
Figure 4-3	Navigating to a TV show: example of interactions between a device and an iPod   360
Figure 4-4	Navigating to a movie: example of interactions between a device and an iPod   362
Table 4-1	Small packet format   343
Table 4-2	Large packet format   344
Table 4-3	RetiPodOptionsForLingo option bits for Lingo 0x04   345
Table 4-4	Simplified IDPS and authentication command sequence   345
Table 4-5	General lingo commands for switching modes   348
Table 4-6	Database category hierarchy (excluding podcasts)   350
Table 4-7	Database category hierarchy for podcasts   351
Table 4-8	Selecting playlists containing video   363
Table 4-9	Typical extended interface commands   364
Table 4-10	Extended Interface lingo command summary   368
Table 4-11	ACK command   372
Table 4-12	GetCurrentPlayingTrackChapterInfo command   373
Table 4-13	ReturnCurrentPlayingTrackChapterInfo command   373
Table 4-14	SetCurrentPlayingTrackChapter command   374
Table 4-15	GetCurrentPlayingTrackChapterPlayStatus command   375
Table 4-16	ReturnCurrentPlayingTrackChapterPlayStatus command   376
Table 4-17	GetCurrentPlayingTrackChapterName command   377
Table 4-18	ReturnCurrentPlayingTrackChapterName command   377
Table 4-19	GetAudiobookSpeed command   378
Table 4-20	ReturnAudiobookSpeed command   378
Table 4-21	Audiobook speed states   379
Table 4-22	SetAudiobookSpeed command to set the speed of the currently playing audiobook   380
Table 4-23	SetAudiobookSpeed command to set the global audiobook speed   380
Table 4-24	GetIndexedPlayingTrackInfo command   381
Table 4-25	Track information type   381
Table 4-26	ReturnIndexedPlayingTrackInfo command for info types 0x00-0x02 and 0x05-0x07   382
Table 4-27	ReturnIndexedPlayingTrackInfo command for info types 0x03-0x04   383
Table 4-28	Track info types and return data   384
Table 4-29	Track Capabilities and Information encoding   384
Table 4-30	Track Release Date encoding   385
Table 4-31	GetArtworkFormats packet   385
Table 4-32	RetArtworkFormats packet   386
Table 4-33	GetTrackArtworkData packet   387
Table 4-34	RetTrackArtworkData packet   388
Table 4-35	ResetDBSelection command   389
Table 4-36	SelectDBRecord command   391
Table 4-37	Database category types for commands   391

Table 4-38	GetNumberCategorizedDBRecords <b>command</b>	392
Table 4-39	ReturnNumberCategorizedDBRecords <b>command</b>	393
Table 4-40	RetrieveCategorizedDatabaseRecords <b>command</b>	394
Table 4-41	ReturnCategorizedDatabaseRecord <b>command</b>	395
Table 4-42	GetPlayStatus <b>command</b>	396
Table 4-43	ReturnPlayStatus <b>command</b>	396
Table 4-44	GetCurrentPlayingTrackIndex <b>command</b>	397
Table 4-45	ReturnCurrentPlayingTrackIndex <b>command</b>	398
Table 4-46	GetIndexedPlayingTrackTitle <b>command</b>	399
Table 4-47	ReturnIndexedPlayingTrackTitle <b>command</b>	399
Table 4-48	GetIndexedPlayingTrackArtistName <b>command</b>	400
Table 4-49	ReturnIndexedPlayingTrackArtistName <b>command</b>	401
Table 4-50	GetIndexedPlayingTrackAlbumName <b>command</b>	401
Table 4-51	ReturnIndexedPlayingTrackAlbumName <b>command</b>	402
Table 4-52	<b>One-byte</b> SetPlayStatusChangeNotification <b>command</b>	403
Table 4-53	<b>Four-byte</b> SetPlayStatusChangeNotification <b>command</b>	403
Table 4-54	<b>One-byte status change event values</b>	404
Table 4-55	<b>Four-byte status change event mask bits</b>	404
Table 4-56	PlayStatusChangeNotification <b>command</b>	405
Table 4-57	<b>Play status change notification codes</b>	405
Table 4-58	PlayCurrentSelection <b>command</b>	407
Table 4-59	PlayControl <b>command</b>	408
Table 4-60	<b>Play control command codes</b>	408
Table 4-61	GetTrackArtworkTimes <b>packet</b>	409
Table 4-62	RetTrackArtworkTimes <b>packet</b>	411
Table 4-63	GetShuffle <b>command</b>	411
Table 4-64	ReturnShuffle <b>command</b>	412
Table 4-65	<b>Shuffle modes</b>	412
Table 4-66	SetShuffle <b>command</b> with Restore on Exit byte	413
Table 4-67	SetShuffle <b>command</b>	414
Table 4-68	GetRepeat <b>command</b>	414
Table 4-69	ReturnRepeat <b>command</b>	415
Table 4-70	<b>Repeat state values</b>	415
Table 4-71	SetRepeat <b>command</b> with Restore on Exit byte	416
Table 4-72	SetRepeat <b>command</b>	416
Table 4-73	SetDisplayImage <b>descriptor command</b> (packet index = 0x0000)	417
Table 4-74	SetDisplayImage <b>data packet</b> (packet index = 0x0001 – 0xNNNN)	418
Table 4-75	<b>Display pixel format codes</b>	420
Table 4-76	<b>2 bpp monochrome pixel intensities</b>	420
Table 4-77	GetMonoDisplayImageLimits <b>command</b>	422
Table 4-78	ReturnMonoDisplayImageLimits <b>command</b>	423
Table 4-79	GetNumPlayingTracks <b>command</b>	423
Table 4-80	ReturnNumPlayingTracks <b>command</b>	424
Table 4-81	SetCurrentPlayingTrack <b>command</b>	425
Table 4-82	SelectSortDBRecord <b>command</b>	426
Table 4-83	<b>Database sort order options</b>	426

Table 4-84	GetColorDisplayImageLimits command	428
Table 4-85	ReturnColorDisplayImageLimits command	428
Table 4-86	ResetDBSelectionHierarchy command	429
Table 4-87	GetDBiTunesInfo command	430
Table 4-88	iTunes database metadata types	430
Table 4-89	RetDBiTunesInfo command	431
Table 4-90	iTunes database metadata information formats	431
Table 4-91	Date/time format	432
Table 4-92	GetUIDTrackInfo command	432
Table 4-93	Track information type bits	433
Table 4-94	RetUIDTrackInfo command	435
Table 4-95	Track information data formats	435
Table 4-96	Capabilities bits	437
Table 4-97	GetDBTrackInfo command	438
Table 4-98	RetDBTrackInfo command	439
Table 4-99	GetPBTrackInfo command	439
Table 4-100	RetPBTrackInfo command	440
Table 4-101	History of the iPod Extended Interface protocol	441

## Appendix A      **Accessory Identification   445**

---

Table A-1	IDPS General lingo commands	446
Table A-2	IDPS process up to authentication	446
Table A-3	IDPS process with authentication and Digital Audio lingo	447

## Appendix B      **Transaction IDs   451**

---

Table B-1	Forms of the General lingo ACK command	452
Table B-2	General lingo ACK packet with transaction ID	453
Table B-3	ACK command error codes	453
Table B-4	General lingo ACK packet with transaction ID and Command Pending status	454
Table B-5	General lingo ACK packet with Command Pending status	455
Table B-6	Default General lingo ACK packet	455
Table B-7	Example of IDPS and authentication	456
Table B-8	Example of entering Remote UI mode	457
Table B-9	Example of getting track artwork data	457
Table B-10	Example of sending notifications	458

## Appendix C      **Multisection Data Transfers   459**

---

Table C-1	Multisection iPodACK packet	460
Table C-2	Multisection DevACK packet	461

**Appendix D iTunes Tagging 463**

---

Figure D-1	iTunes tagging feature data flows	464
Figure D-2	iTunes tagging element sequence	468
Figure D-3	RDS 3A group for iTunes tagging	468
Figure D-4	RDS application group for iTunes tagging	469
Figure D-5	Event Control element	470
Figure D-6	Element decryption process	471
Table D-1	RT+ content types	466
Table D-2	FM tag element types	467
Table D-3	Outer mask lookup table	472
Table D-4	Permutation lookup table, upper bits	472
Table D-5	Permutation lookup table, lower bits	473
Table D-6	Element type lookup table	474
Table D-7	HD UFID data ID types	476
Table D-8	Plist fields written to the iPod	479
Table D-9	Tagging feature user interface implementation	482
Table D-10	Tagging feature UI text messages	483
Table D-11	Radio tagging command sequence	484
Listing D-1	Pseudo-random number generator	475

**Appendix E Nike + iPod Cardio Equipment System 495**

---

Figure E-1	Sample user experience flow chart	509
Table E-1	Lingo version support	496
Table E-2	Sample identification process 1	497
Table E-3	Sample identification process 2	498
Table E-4	Writing workout data to the iPod	499
Table E-5	XML element usage	502
Table E-6	Approved user interface messages	508
Table E-7	Cardio equipment command sequence using IDPS	510

**Appendix F Historical Information 517**

---

Table F-1	Past iPod models, firmware, and lingo versions, table 1	517
Table F-2	Past iPod models, firmware, and lingo versions, table 2	519
Table F-3	BeginRecord packet	521
Table F-4	EndRecord packet	521
Table F-5	BeginPlayback packet	522
Table F-6	EndPlayback packet	522
Table F-7	Identify packet	523
Table F-8	Identify packet for high-power devices	524
Table F-9	Power option bits	524
Table F-10	USB Host Control lingo command summary	525
Table F-11	Select Host Control command timings	525

Table F-12	USB Host Control lingo command history	526
Table F-13	ACK packet	526
Table F-14	GetUSBPowerState packet	526
Table F-15	RetUSBPowerState packet	527
Table F-16	SetUSBPowerState packet	527
Table F-17	RequestProtocolVersion command	528
Table F-18	ReturnProtocolVersion command	529
Table F-19	RequestiPodName command	530
Table F-20	ReturniPodName command	530
Table F-21	Legacy accessory authentication	531
Table F-22	UART accessory identification with non-IDPS iPods	532
Table F-23	USB or BT accessory identification with non-IDPS iPods	534
Table F-24	Non-IDPS identification of lingo 0x00+0x04	535
Table F-25	Non-IDPS identification of lingo 0x00+0x02+0x03	539
Table F-26	Non-IDPS identification of lingo 0x00+0x02+0x03+0x0A	542
Table F-27	Non-IDPS radio tagging command sequence	545
Table F-28	Non-IDPS cardio equipment command sequence	548



# Introduction

---

**NOTICE OF PROPRIETARY PROPERTY:** THE INFORMATION CONTAINED HEREIN IS THE PROPRIETARY PROPERTY OF APPLE INC. THE POSSESSOR AGREES TO THE FOLLOWING: (I) TO MAINTAIN THIS DOCUMENT IN CONFIDENCE, (II) NOT TO REPRODUCE OR COPY IT, (III) NOT TO REVEAL OR PUBLISH IT IN WHOLE OR IN PART, (IV) ALL RIGHTS RESERVED.

ACCESS TO THIS DOCUMENT AND THE INFORMATION CONTAINED THEREIN IS GOVERNED BY THE TERMS OF THE IPOD CONNECTOR USE LICENSE AGREEMENT AND/OR THE IPOD TECHNOLOGY EVALUATION LICENSE AGREEMENT. ALL OTHER USE SHALL BE AT APPLE'S SOLE DISCRETION.

This document specifies the electrical and software interfaces to the Apple iPod that are available to accessory devices. The iPod models covered by this specification are shown in [Table I-1](#) (page 30).

**Note:** This document does not apply to the first- and second-generation iPods, nor to the iPod shuffle.












To determine if a command or feature is supported in a particular iPod model or firmware release, see [Table 1-4](#) (page 38) and [Table 1-5](#) (page 39).













**Note:** The iPhone and iPod touch share the same iPod Accessory Protocol (iAP) interface except where specified otherwise in this document. All accessories for the iPhone must meet additional electrical and RF design requirements. These requirements are described in the iPhone license agreement and in Apple's *iPod/iPhone Accessory Testing and Certification Specification*.











The minimum iPod System Software versions supported by this specification are:



- Version 2.0 of the third-generation (3G) iPod.
- Version 3.0 of the fourth-generation (4G) iPod.
- Version 1.0 of the iPod mini, fourth-generation iPod (color display), iPod nano, fifth-generation iPod (5G), second-generation iPod nano, iPod classic, and iPod 3G nano.
- Version 1.0.2 of the 4G nano.
- Version 1.1 of the iPhone, iPhone 3G, and iPod touch.
- Version 2.1.1 of the 2G iPod touch.
- Version 3.0.0 of the iPhone 3GS.

**Table I-1** iPod and iPhone products

3G iPod		
	Product name: iPod (3rd generation)	Shipped: 04/2003
	Compatibility icons: <div><div>iPod 3rd generation 10GB 15GB 20GB</div><div>iPod 3rd generation 30GB 40GB</div></div>	
	Connectivity: Dock connector, headphone jack	
iPod mini		
	Product name: iPod mini	Shipped: 01/2004
	Compatibility icon: <div>iPod mini 4GB 6GB</div>	
	Connectivity: Dock connector, headphone jack	
4G iPod		
<div></div> <div></div>	Product names: iPod (4th generation), iPod photo, iPod photo (2nd generation), iPod 4th generation (color display)	Shipped: 07/2004, 09/2004, 02/2005, 06/2005
	Compatibility icons: <div><div>iPod 4th generation 20GB</div><div>iPod 4th generation 40GB</div><div>iPod 4th generation (color display) 20GB 30GB</div><div>iPod 4th generation (color display) 40GB 60GB</div></div>	
	Connectivity: Dock connector, headphone jack	

iPod nano		
	<i>Product name:</i> iPod nano	<i>Shipped:</i> 09/2005
	<i>Compatibility icon:</i>  iPod nano 1st generation 1GB 2GB 4GB	
	<i>Connectivity:</i> Dock connector, headphone jack	
5G iPod		
	<i>Product name:</i> iPod with video	<i>Shipped:</i> 10/2005
	<i>Compatibility icons:</i>  iPod 5th generation (video) 30GB  iPod 5th generation (video) 60GB 80GB	
	<i>Connectivity:</i> Dock connector, headphone jack	
2G nano		
	<i>Product name:</i> iPod nano (2nd generation)	<i>Shipped:</i> 09/2006
	<i>Compatibility icon:</i>  iPod nano 2nd generation (aluminum) 2GB 4GB 8GB	
	<i>Connectivity:</i> Dock connector, headphone jack	
iPod classic, 120 GB classic, 160 GB classic		
	<i>Product names:</i> iPod classic, iPod classic (120GB), iPod classic (160GB)	<i>Shipped:</i> 09/2007, 09/2008, 9/2009
	<i>Compatibility icons:</i>  iPod classic 80GB  iPod classic 120GB  iPod classic 160GB (2007)  iPod classic 120GB 160GB (2009)	
	<i>Connectivity:</i> Dock connector, headphone jack	

3G nano		
	Product name: iPod nano (3rd generation)	Shipped: 09/2007
	Compatibility icon:  iPod nano 3rd generation (video) 4GB 8GB	
	Connectivity: Dock connector, headphone jack	
4G nano		
	Product name: iPod nano (4th generation)	Shipped: 09/2008
	Compatibility icon:  iPod nano 4th generation (video) 8GB 16GB	
	Connectivity: Dock connector, microphone/headphone jack	
iPhone, iPhone 3G, iPhone 3GS		
	Product names: iPhone, iPhone 3G, iPhone 3GS	Shipped: 06/2007, 06/2008, 6/2009
	Compatibility icons:  iPhone 4GB 8GB 16GB  iPhone 3G 8GB 16GB  iPhone 3GS 16GB 32GB	
	Connectivity: GSM, WiFi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	
iPod touch, 2G touch		
	Product names: iPod touch, iPod touch (2nd generation)	Shipped: 09/2007, 02/2008, 09/2008, 09/2009
	Compatibility icons:  iPod touch 8GB 16GB 32GB  iPod touch 2nd generation 8GB 16GB 32GB 64GB	
	Connectivity: WiFi, Bluetooth, multi-touch display, dock connector, microphone/headphone jack	

5G nano		
	Product name: iPod nano (5th generation)	Shipped: 09/2009
	Compatibility icon:  iPod nano 5th generation (video camera) 8GB 16GB	
	Connectivity: Dock connector, microphone/headphone jack	

## Organization of This Document

The specifications in this document are arranged in four chapters:

- ["Protocol Features and Availability"](#) (page 35) gives an overview of the General and accessory lingo and their availability.
- ["The Protocol Core and the General Lingo"](#) (page 47) gives an overview of the iAP and describes the General Lingo, which all accessories must support.
- ["Accessory Lingo"](#) (page 145) describes the various device-specific accessory lingo that are part of the iAP and their commands.
- ["The Extended Interface Protocol"](#) (page 341) describes iAP Lingo 0x04, which allows the user interface of the iPod to be translated to other environments.

Several appendixes provide additional information for both hardware and software designers:

- ["Accessory Identification"](#) (page 445) specifies the Identify Device Preferences and Settings (IDPS) process which accessories identify themselves.
- ["Transaction IDs"](#) (page 451) describes the transaction ID parameters used by the Location lingo, the IDPS process, and multisection data transfers.
- ["Multisection Data Transfers"](#) (page 459) tells how to transfer amounts of data larger than the data receiver's maximum incoming packet size.
- ["iTunes Tagging"](#) (page 463) describes the iTunes tagging feature for HD and FM radio accessories.
- ["Nike + iPod Cardio Equipment System"](#) (page 495) describes the use of the Sports and Storage lingo with Nike + iPod cardio equipment.
- ["Historical Information"](#) (page 517) provides specifications for past iPod models and iAP protocols.

At the end of this document are a glossary of terms and a document revision history.

## Specification Terms

Parts of this document contain specification requirements that are incorporated by reference into legal agreements between Apple Inc. and its licensees. The use of the words “must,” “should,” “may,” and “reserved” in these specifications have the following meanings:

- “Must” means that the specification is an absolute requirement.
- “Must not” means that the specification is an absolute prohibition.
- “Should” means that there may be valid reasons in particular circumstances to ignore the specification, but their full implications must be understood and carefully weighed before choosing to do so.
- “Should not” means that there may be valid reasons in particular circumstances that make the specified action or feature acceptable, but their full implications must be understood and carefully weighed before choosing to include it.
- “May” means that the indicated action or feature does not contravene this specification.

## See Also

For further information, refer to the latest revisions of these additional documents:

- *IEEE 1394a Specification*
- *USB 2.0 High Speed Specification*
- *USB Device Class Definition for Audio Devices*
- *USB Device Class Definition for Audio Data Formats*
- *USB Device Class Definition for Human Interface Devices (HID)*
- *United States RBDS Standard, NRSC-4-A*

# Protocol Features and Availability

[Table 1-1](#) (page 35) lists the current firmware and iPhone OS versions for each iPod and iPhone model.

**Table 1-1** Current firmware and OS versions in iPod and iPhone models

Model	Firmware version	Firmware revision date	iPhone OS version	OS revision date
iPod 3G <sup>1</sup>	2.3	02/2005		
iPod mini	1.4.1	01/2006		
4G iPod	3.1.1	01/2006		
4G iPod (color display)	1.2.1	01/2006		
1G nano	1.3.1	03/2007		
iPod 5G	1.3	03/2008		
2G nano	1.1.3	05/2007		
iPod classic	1.1.2	05/2008		
3G nano	1.1.3	07/2008		
iPod touch			1.1.5	07/2008
			2.2.1	01/2009
			3.1.2	10/2009
iPhone			3.1.2	10/2009
iPhone 3G			3.1.2	10/2009
4G nano	1.0.4	08/2009		
120 GB classic	2.0.3	09/2009		
2G touch			2.2.1	01/2009
			3.1.2	10/2009
iPhone 3GS			3.1.2	10/2009
160 GB classic	2.0.3	09/2009		
5G nano	1.0.1	09/2009		

Table 1-2 (page 36) and Table 1-3 (page 37) list the lingo protocol versions for each iPod model's most recent firmware and OS version. In these tables, "NL" indicates that the given lingo was not implemented in the specified model loaded with the specified firmware. "NV" indicates that although the lingo was implemented, its protocol version could not be read from the iPod.

**Table 1-2** Most recent iPod models, firmware, and lingo versions, Table 1

iPod models	3G <sup>1</sup>	mini	4G	photo; 4G (color display)	1G nano	5G	2G nano	classic	3G nano	1G touch
Most recent Firmware	2.3	1.4.1	3.1.1	1.2.1	1.3.1	1.3	1.1.3	1.1.2	1.1.3	3.1.2
Last Revision Date	02/ 2005	01/ 2006	01/ 2006	01/ 2006	03/ 2007	03/ 2008	05/ 2007	05/ 2008	07/ 2008	10/ 2009
Most recent lingo protocol versions										
General 0x00	NV	1.02	1.02	1.02	1.05	1.06	1.06	1.07	1.07	1.09
Microphone 0x01	NV	NL	1.00	1.00	NL	1.01	1.01	1.01	1.01	1.02
Simple Remote 0x02	NV	1.00	1.00	1.00	1.02	1.02	1.02	1.02	1.02	1.02
Display Remote 0x03	NL	1.01	1.01	1.01	1.05	1.05	1.04	1.05	1.05	1.05
Extended Interface 0x04	1.02	1.05	1.05	1.09	1.11	1.12	1.10	1.13	1.13	1.12
Accessory Power 0x05	NL	1.00	1.00	1.00	1.01	1.01	1.01	1.01	1.01	1.01
USB Host Control 0x06	NL	NL	NL	1.00	NL	1.00	NL	NL	NL	NL
RF Tuner 0x07	NL	NL	NL	NL	1.00	1.00	1.00	1.00	1.00	NL
Accessory Equalizer 0x08	NL	NL	NL	NL	1.00	1.00	1.00	1.00	1.00	1.00
Sports 0x09	NL	NL	NL	NL	NL	NL	NL	NL	1.01	NL
Digital Audio 0x0A	NL	NL	NL	NL	1.01	1.01	1.02	1.03	1.03	1.02
Storage 0x0C	NL	NL	NL	NL	NL	NL	NL	NL	1.02	1.02
Location 0x0E	NL	NL	NL	NL	NL	NL	NL	NL	NL	1.00



<sup>1</sup> Supporting the 3G iPod requires special design considerations. See “Interfacing With the 3G iPod” in *iPod/iPhone Hardware Specifications*.

**Table 1-3** Most recent iPod/iPhone models, firmware, and lingo versions, Table 2

iPod/iPhone models	iPhone	iPhone 3G	4G nano	120 GB classic	2G touch	iPhone 3GS	160 GB classic	5G nano
Most recent Firmware	3.1.2	3.1.2	1.0.4	2.0.1	3.1.2	3.1.2	2.0.3	1.0.1
Last Revision Date	10/2009	10/2009	08/2009	01/2009	10/2009	10/2009	09/2009	09/2009
Most recent lingo protocol versions								
General 0x00	1.09	1.09	1.08	1.08	1.09	1.09	1.08	1.08
Microphone 0x01	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02
Simple Remote 0x02	1.02	1.02	1.04	1.03	1.02	1.02	1.03	1.04
Display Remote 0x03	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05
Extended Interface 0x04	1.12	1.12	1.14	1.13	1.12	1.12	1.13	1.14
Accessory Power 0x05	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01
USB Host Control 0x06	NL	NL	NL	NL	NL	NL	NL	NL
RF Tuner 0x07	NL	NL	1.01	1.00	NL	NL	1.00	1.01
Accessory Equalizer 0x08	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Sports 0x09	NL	NL	1.01	NL	1.01	1.01	NL	1.01
Digital Audio 0x0A	1.02	1.02	1.03	1.03	1.02	1.02	1.03	1.03
Storage 0x0C	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02
Location 0x0E	1.00	1.00	NL	NL	1.00	1.00	NL	NL

The accessory should determine what features an attached iPod supports by sending the General lingo command 0x4B, `GetiPodOptionsForLingo`. If the iPod does not support that command, the accessory must extract the lingo version number from the iPod. This protocol version information can be used to determine which features the connected iPod supports. See ["Command 0x0F: RequestLingoProtocolVersion"](#) (page 79) for information about the `RequestLingoProtocolVersion` command.

For past firmware versions, see [Table F-1](#) (page 517). [Table 1-4](#) (page 38) lists hardware and software features outside these protocols.

## General iPod Features

[Table 1-4](#) (page 38) and [Table 1-5](#) (page 39) list iPod/iPhone hardware and software features that are not part of specific lingoes. The numbers in the tables show the firmware versions in which each of these features was introduced.

**Table 1-4** Features supported by specific iPod firmware and OS versions, Table 1

Features	Software versions									
	3G	mini	4G	photo; 4G (color display)	nano	5G	2G nano	classic	3G nano	1G touch
Serial autobaud on framing errors	—	1.4.0	3.1.0	1.1.0	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.1.0
Display notification on unsupported iAP device attach	—	—	—	—	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.1.0
iPod detects detach for all valid R <sub>ID</sub> values	—	1.4.0	3.1.0	1.2.0	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.1.0
Authentication Version 1.00	—	—	—	—	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.1.0
Authentication Version 2.00	—	—	—	—	1.2.0	1.2.0	1.0.0	1.0.0	1.0.0	1.1.0
USB audio	—	—	—	—	1.2.0 <sup>1</sup>	1.2.0	1.1.2	1.0.0	1.0.0	1.1.0
Video browsing <sup>2</sup>	—	—	—	—	—	1.2.0	—	1.0.3	1.0.3	2.0.0
iTunes tagging	—	—	—	—	—	1.2.3	—	1.0.0	1.0.0	2.1.0
Headphone remote <sup>3</sup> and mic system	—	—	—	—	—	—	—	—	—	—
Nike + iPod cardio equipment	—	—	—	—	—	—	—	—	1.1.2	—
Communication with iPhone OS applications	—	—	—	—	—	—	—	—	—	3.0.0

**Table 1-5** Features supported by specific iPod/iPhone firmware and OS versions, Table 2

Features	Software versions							
	iPhone	iPhone 3G	4G nano	120 GB classic	2G touch	iPhone 3GS	160 GB classic	5G nano
Serial autobaud on framing errors	2.1.0	2.1.0	1.02	2.0.0	2.1.1	3.0.0	2.0.3	1.0.1
Display notification on unsupported iAP device attach	2.1.0	2.1.0	1.02	2.0.0	2.1.1	3.0.0	2.0.3	1.0.1
iPod detects detach for all valid R <sub>ID</sub> values	2.1.0	2.1.0	1.02	2.0.0	2.1.1	3.0.0	2.0.3	1.0.1
Authentication Version 1.00	2.1.0	2.1.0	1.02	2.0.0	2.1.1	3.0.0	2.0.3	1.0.1
Authentication Version 2.00	2.1.0	2.1.0	1.02	2.0.0	2.1.1	3.0.0	2.0.3	1.0.1
USB audio	2.1.0	2.1.0	1.02	2.0.0	2.1.1	3.0.0	2.0.3	1.0.1
Video browsing <sup>2</sup>	2.1.0	2.1.0	1.02	2.0.0	2.1.1	3.0.0	2.0.3	1.0.1
iTunes tagging	2.1.0	2.1.0	1.02	2.0.0	2.1.1	3.0.0	2.0.3	1.0.1
Headphone remote <sup>3</sup> and mic system	—	—	1.02	2.0.0	2.1.1	3.0.0	2.0.3	1.0.1
Nike + iPod cardio equipment	—	—	1.02	—	2.1.1	3.0.0	—	1.0.1
Communication with iPhone OS applications	3.0.0	3.0.0	—	—	3.0.0	3.0.0	—	—

<sup>1</sup> Because version 1.00 of Lingo 0x0A contained a bug that was corrected in version 1.0.1, accessories should attempt Digital Audio only with iPods whose Lingo 0x0A version is higher than or equal to 1.0.1. See [Table 3-218](#) (page 294).

<sup>2</sup> With iPod models that store and display video content (in addition to music), an accessory device may choose to navigate the iPod's hierarchy of stored videos. For details, see ["Video Browsing"](#) (page 358).

<sup>3</sup> In the headphone remote and mic system, remote control from the headphone is limited to volume +/- support.

## Accessory Power Policy

All accessories must comply with the power states of the iPod as specified in this section.

Accessories may draw 5 mA plus any current required by the Apple Authentication Coprocessor, up to the peak current drain specified in Apple's *iPod Authentication Coprocessor 2.0B Specification, Release R4*. They may draw this total current throughout the Authentication process, which begins at the rising edge of the Accessory Power line (pin 13 of the 30-pin connector) and ends 500 ms after the iPod sends the accessory an `AckDevAuthenticationStatus` command with a status of 0x00, Success (OK).

Accessories that declare Low Power during the identification process are limited to drawing 5 mA peak current from the Accessory Power line at all times except during authentication.

Accessories may register for software-controlled intermittent high power during the identification process. If the iPod sends an `AckDevAuthenticationStatus` command with a status of 0x00, they can draw up to 100 mA peak current from the Accessory Power line upon receipt of any one or more of the following iAP commands:

- **Microphone lingo:**
  - `iPodModeChange`: Begin audio recording mode
  - `iPodModeChange`: Begin audio playback mode
- **Accessory Power lingo:**
  - `BeginHighPower`
- **RF Tuner lingo:**
  - `SetTunerCtrl`: Turn RF tuner device power on
- **General lingo (communicating with iPhone OS 3.0 applications):**
  - `OpenDataSessionForProtocol`
- **Location lingo:**
  - `SetDevControl`: Accessory GPS radio power, Power on

Accessories must reduce their power consumption to no more than 5 mA within 1 second after all the commands that set high power are canceled by one or more of the following commands:

- **Microphone lingo:**
  - `iPodModeChange`: End audio recording mode
  - `iPodModeChange`: End audio playback mode
- **Accessory Power lingo:**
  - `EndHighPower`
- **RF Tuner lingo:**
  - `SetTunerCtrl`: Turn RF tuner device power off
- **General lingo (communicating with iPhone OS 3.0 applications):**
  - `CloseDataSession`

- Location lingo:
  - `SetDevControl`: Accessory GPS radio power, Power off

At all other times, accessories must comply with the 5 mA Low Power accessory rules.

Accessories may register for constant high power mode during the identification process if they power the iPod as a charger or battery pack using the method defined in “Supplying USB Power” in appendix “iPod Power States and Accessory Power” in *iPod/iPhone Hardware Specifications*. In this mode they may draw 100 mA continuously after the iPod sends an `AckDevAuthenticationStatus` command with a status of 0x00.

## Video Output Preferences

The iAP General lingo includes commands that let automotive head units and other accessories determine if the iPod supports video output, and if so, to control several video output preferences. All iPod models let an accessory control the video screen configuration and format. Newer iPods also let the user control the video output connection type, aspect ratio, closed caption enabling, and subtitle behavior, as shown in [Table 1-6](#) (page 41).

**Table 1-6** Video output capabilities

Capability	5G iPod	iPod classic; 3G, 4G, and 5G nano	iPod touch, iPhone	2G touch	iPhone 3G, iPhone 3GS
Widescreen signaling	No	Yes	No		
Composite (interlaced) output	Yes	Yes	Yes		
S-video (interlaced) output	Yes	Yes	Yes		
Component (interlaced) output	No	No	Yes	No	
Component (progressive) output	No	Yes	No	Yes	
4:3 fullscreen aspect ratio	Yes	Yes	Yes		
16:9 widescreen aspect ratio	No	Yes	Yes		
Closed captioning	No	Yes	Yes		
Subtitle display	No	Yes	No		
Display resolution	640 x 480	NTSC: 720 x 480; PAL: 720 x 576			

An accessory can determine if the iPod supports video output by sending a `GetiPodOptionsForLingo` command. If the iPod does not support that command, the accessory may send a `GetiPodOptions` command. If the iPod returns an `ACK` command with a Bad Parameter status, or sends a `RetiPodOptions` command with the Video Option bit (bit 0) clear, the iPod does not support video. If the video option bit is set, the iPod at least supports video output, screen configuration, and video format preferences. The accessory can

determine if the iPod supports other preferences by sending a `GetiPodPreferences` command for each preference. If the iPod returns an `ACK` command with `Bad Parameter` status, that preference is not supported (see "[Command 0x29: GetiPodPreferences](#)" (page 102)).

## Character Encoding

The iTunes application and all iPods and iPhones use Unicode UTF-8 encoding for tags and metadata. UTF-8 is compatible with the ASCII character set and supports other languages, including Chinese, Japanese, and Korean. It specifies unique encodings to represent all its supported character glyphs including the Roman alphabet and Han, Kanji, Kana, and Hangul characters. The UTF-8 format uses 1 to 4 bytes to represent each encoding; this means that a 5-character word can require from 5 to 20 bytes of storage. More information on Unicode and UTF-8 encoding can be found at <http://www.unicode.org/>.

Every accessory should decode UTF-8 characters. If the accessory is not capable of displaying certain Unicode character glyphs, a substitute glyph (such as an open square) should be displayed instead. Accessories should make provision to decode and render the curly apostrophe character (ASCII 0xD5), which occurs in all iTunes text.

## Accessory Control of the iPod touch and iPhone

Accessory developers must take into account these current limitations on the ability of accessories to control an iPod touch or iPhone.

- An accessory cannot navigate the home screen of an iPod touch or iPhone.
- An accessory cannot unlock an iPod touch or iPhone screen once it is locked.
- When the iPod touch or iPhone screen is locked, most iAP commands continue to operate, but some Simple Remote lingo commands (such as the menu, select, and arrow button controls) produce no visible effect.
- Accessories are responsible for adjusting video out settings if they are not entering or using the Extended Interface mode. If it needs to display video outside the display on an iPhone or iPod touch, the accessory should set the video out preference to On, or to Ask if it wants the user to be prompted. In Remote UI mode, the Ask preference defaults to On and no prompt appears.

## Accessory Communication With iPhone OS Applications

This section describes the communication process between accessories and the iPhone OS and specifies the iAP commands that an accessory can use to open and maintain communication with an iPhone OS application. For information about iAP command formats, see "[Command Packet Formats](#)" (page 58).

**Note:** Before it can communicate with any iPhone OS application, an accessory must be identified through the process described in "Accessory Identification" (page 445) and must authenticate itself using Authentication 2.0B, as described in "Authentication" (page 52). Using IDPS also requires enabling transaction IDs in iAP commands, as described in "Transaction IDs" (page 451).

The IDPS process lets an iPhone or iPod touch match its applications with its attached accessories, making it possible to open communication between them. Once an accessory has been identified and authenticated, the process described below can create a route for two-way data transmission between it and compatible iPhone OS applications.

The communication process between an iPhone or iPod touch accessory and an iPhone OS application requires both accessory iAP commands and application programming. The iAP side of the process is described in this section. For iPhone OS programming information, visit [developer.apple.com/iphone/program](http://developer.apple.com/iphone/program).

## Setting Up a Communication Session

To communicate with an iPhone OS application, an accessory must first expose certain protocol identifiers by sending `protocolString` values to the iPod or iPhone; see [Table 2-78](#) (page 114). This information is used to establish communication channels to the application. The accessory must also send an Apple-assigned `BundleSeedIDString` value, as shown in [Table 2-79](#) (page 115). This string identifies the accessory's preferred application. The Bundle Seed ID string is assigned to the application developer through the iPhone Developer Program, as part of the application development process. For further information, see "Managing Devices" in *iPhone Development Guide*, available at <http://developer.apple.com/iphone>. The accessory sends the `BundleSeedIDString` value as part of its startup identification, as described in "Accessory Identification" (page 445).

The design and maintenance of communication protocols between accessories and applications are entirely the responsibility of the developers of these products. Apple makes no attempt to secure protocol name space or provide for communication security between accessories and applications. Protocol names must be in reverse-DNS format and must be associated with domains that are registered with the Internet Corporation for Assigned Names and Numbers (ICANN), so that Apple can rely on each protocol having a unique owner. Apple does not require a developer to be the owner of the domain name used, but will require (by self-certification) that the developer has permission to use the domain name for the protocol. For suggestions on protocol design, see "Protocol Design Hints" (page 45).

When an iPhone OS application requests access to an accessory, the iPhone or iPod touch sends it an `OpenDataSessionForProtocol` command. As a part of this notification, `sessionID` and `protocolIndex` values are sent to the accessory, so it can identify incoming iAP commands for this session. A session ID is included in all iAP data transfer commands, so that multiple sessions may run concurrently. The accessory should accept the connection with a `DevACK` response.

When the application has finished with the accessory, or the iPhone or iPod touch has determined that communication is no longer needed, a `CloseDataSession` command is sent to the accessory. The accessory must cease communication through the `sessionID` that was just closed. At this time the accessory can implement any power-saving features it may have.

## Data Flow from the iPod or iPhone

---

The iAP command `iPodDataTransfer` is used to communicate information to the accessory from the application and `DevDataTransfer` to communicate information to the application from the accessory. Both iAP commands contain a `sessionID` parameter to identify which channel they are traveling through. The message delivery order is first in, first out. End-to-end delivery of data is normally reliable, but not absolutely guaranteed, so developer protocols must be designed to recover from loss of data. For information about iAP transport links, see “Protocol Transport Links” in *iPod/iPhone Hardware Specifications*.

During a communication session, the following rules apply to `iPodDataTransfer` commands:

- The iPod will send an initial `iPodDataTransfer` packet when an application has placed data into the accessory’s input queue.
- Upon receiving an acknowledgment of an `iPodDataTransfer` command, the iPod may immediately send a new command containing available data for any session ID.
- The accessory’s acknowledgment of an `iPodDataTransfer` command with Success status (0x00) must signify that the packet has been received, it has been moved out of the accessory’s lowest-level receive queue, and that the accessory is ready to receive another packet.
- The only circumstance under which an accessory may acknowledge an `iPodDataTransfer` command with a nonzero status is if no prior `OpenDataSessionForProtocol` command has established the command’s session ID. In this case it must send a Bad Parameter (0x04) status, and the iPod may close the session.
- If an `iPodDataTransfer` command is acknowledged with an error status, the iPod may optionally purge its outbound packet queue of all pending `iPodDataTransfer` packets for that session ID. The iPod will not otherwise discard any data in a session.
- If an accessory’s data receive queue cannot accept another packet, then the accessory must not acknowledge the `iPodDataTransfer` command until packet acceptance is restored. This may cause the iPod to retry the `iPodDataTransfer` command up to ten times. If the iPod receives no acknowledgment after the tenth retry, it may close the session.
- If a 500 ms timeout occurs before an `iPodDataTransfer` command is acknowledged, the iPod will resend the same packet with the same transaction ID.

During a communication session, the following rules apply to `DevDataTransfer` commands:

- The accessory should send a `DevDataTransfer` command whenever it has data it wants to stream to an application with an open session.
- The accessory must not send another `DevDataTransfer` packet until the previous one has been acknowledged (as successful or not) or a 500 ms timeout has expired.
- Upon receiving an acknowledgment of a `DevDataTransfer` command, the accessory may immediately send a new command containing available data for any session ID.
- If a 500 ms timeout occurs before a `DevDataTransfer` command is acknowledged, the accessory may resend the same packet with the same transaction ID.
- If the accessory receives an `iPodNotification` command for flow control, it must not send any packets (including retries of `DevDataTransfer` packets) during the specified waiting time. For information about iPod notification commands, see [iPod Event Notifications](#) (page 449).



## Matching Accessories With Applications

---

The iPhone or iPod touch matches applications and accessories by using the preferences and protocol names communicated by the accessory during its IDPS session. If the accessory names multiple protocols, it will be deemed compatible with an application if the application supports at least one of them. In addition, the accessory's `BundleSeedIDString` value is used to identify preferred or default applications.

A `BundleSeedIDString` is required to properly pair the accessory to its preferred or default application on the iPhone OS device. The application must be submitted to Apple as part of the accessory certification process. Apple assigns a Bundle seed ID value to the application developer, who provides this information to the accessory developer for inclusion in the accessory's firmware.

## Communication Protocol Design Hints

---

The owner or creator of a protocol for communication between accessories and iPhone OS applications must define the preferences that are required or optional. Each accessory and application must then negotiate their level of compatibility directly. Apple does not referee protocols, and protocol creators must ensure that their accessories and applications can verify that they support the same features. The following suggestions can help with protocol design:

- Ensure that the accessory can always resynchronize its data stream in the event of an error.
- The round-trip latency of commands may vary; do not draw inferences from propagation times. There is no guarantee of delivery timing over the communication link.
- After a connection has been established, clearly establish which end speaks first.
- Begin communication with an exchange of meta-information, such as confirmation of the protocol and version. Give both ends the opportunity to declare that they cannot proceed due to protocol incompatibilities.
- Ensure that the transaction order is clear and that both ends know their position in it. Packets within each stream are guaranteed to be ordered but not free of gaps between packets, so ensure that the stream can be resynchronized after a gap.
- Ensure that responses can always be matched to their corresponding requests. The simplest forms of reliable communication are call and response. Interleaved communication streams are more complicated and less desirable.
- Establish an unambiguous indicator of the start of each packet.
- Add security where needed, but recognize that it slows down and complicates each transaction.
- Ensure that the other end of each transaction has both received and understood the last packet.
- Ensure that each packet is received intact and error free by adding a CRC value, checksum, or the like.
- Ensure that incoming data can be cached until it can be processed.
- Try to make the protocol extensible for future needs.

## Communication iAP Commands

The General lingo commands that support iPhone OS applications communicating with accessories are listed in [Table 1-7](#) (page 46) and documented in detail in "[Lingo 0x00: General Lingo](#)" (page 60). All commands are protocol version 1.09 and require authentication 2.0B.

**Table 1-7** Accessory communication commands

CmdID	Name	Direction	Payload:bytes
0x3F	OpenDataSession-ForProtocol	iPod to Dev	{transID;2, sessionID;2, protocolIndex;1}
0x40	CloseDataSession	iPod to Dev	{transID;2, sessionID;2}
0x41	DevACK	Dev to iPod	{transID;2, ackStatus;1, cmdID;1}
0x42	DevDataTransfer	Dev to iPod	{transID;2, sessionID;2, data;<var>}
0x43	iPodDataTransfer	iPod to Dev	{transID;2, sessionID;2, data;<var>}
0x4A	iPodNotification	iPod to Dev	{transID;2, notificationType;1, waitTime;4, overflowTransID;2}

**Note:** The transID parameters in the foregoing table are described in "[Transaction IDs](#)" (page 451).

# The Protocol Core and the General Lingo

This chapter covers the basics of the iPod Accessory Protocol (iAP), including accessory identification and authentication. It also describes the packet format used for iAP command and gives detailed descriptions of the commands included in the General lingo.

The iAP is used for both directions of a link, so every device must implement both sending and receiving capabilities. It should be possible to determine the direction (device to iPod or iPod to device) of a packet only from its contents. This means that no packet is valid for sending from both the iPod and the device.

All devices must be able to handle variable-length packets. An accessory must be able to accept a packet with extra data and ignore the unexpected bytes. At a minimum, the device must not lose sync to the packet signaling.

Most command packets generate a response, either an `ACK` command or a packet with data answering a query. Accessories should wait for the response before querying to see if the original request has been processed by the iPod. In the Display Remote lingo, for example, when an accessory sends a `GetiPodStateInfo` command it should wait for the iPod to return the corresponding `RetiPodStateInfo` (or `ACK`) command before sending a subsequent `GetPlayStatus` command.

**Note:** Unless otherwise stated in this specification, an accessory should wait 3 seconds for the iPod to respond before retrying a command. The absolute minimum for any accessory-generated retry is 500 ms for all lingoes except the Extended Interface lingo, in which the minimum is 1 second. Some special timeout rules apply to the IDPS process; see "[Accessory Identification](#)" (page 445).

## Reserved Commands and Data

From time to time an iPod or iPhone may send an attached accessory a command that is reserved and not documented in this specification, or a documented command that has reserved data in its payload. Accessories must follow these rules when handling such reserved commands or data:

- When a data field is marked "Reserved" in this specification, accessories writing to it must set it to 0 and accessories reading it must ignore its value. The only exception is byte 9 of the `RetTunerCaps` packet shown in [Table 3-99](#) (page 226), which must be set to 0x01.
- If an accessory receives a command documented in this specification that passes reserved data, it must acknowledge successful receipt of the command, extract the data in it that is documented, and ignore the reserved data.
- If an accessory receives a command not documented in this specification, it should ignore it. Alternatively, it may return a General lingo `ACK` command with a value of 0x04 (Bad Parameter) to prevent the iPod from waiting for a timeout period to receive a reply.

## Device Signaling and Initialization

When attached, the accessory must detect the iPod and initiate the identification process. This section describes the initialization process for both the UART serial port link and the USB port link.

### Packet Signaling and Initialization Using the UART Serial Port Link

When using the UART serial port link, the identification process starts when the accessory detects accessory power. The accessory must wait at least 80 ms and then transmit a sync byte. After sending the sync byte, the accessory must wait another 20 ms before starting the IDPS identification process.

**IMPORTANT:** To identify an accessory to an attached iPod or iPhone, before authentication, new accessory designs must use the Identify Device Preferences and Settings (IDPS) process described in Appendix B, "Accessory Identification" (page 445). This process ensures their compatibility with future firmware. Existing accessory designs may continue to send the iAP General lingo command 0x13, `IdentifyDeviceLingoes`, identifying the lingoes they support and requesting immediate authentication. Accessories that need to support the 3G iPod must use the command described in "General Lingo Command 0x01: Identify (Deprecated)" (page 523).

The IDPS accessory identification process begins when the accessory sends "Command 0x38: StartIDPS" (page 108) to the attached iPod. If the iPod does not respond, the accessory may resend `StartIDPS` command as long as iPod Detect (pin 30) is low and Accessory Power (pin 13) is high; see "Hardware Interfaces" in *iPod/iPhone Hardware Specifications*. The accessory must not retry IDPS identification more often than once per second. For 3G iPod support, see "Interfacing With the 3G iPod" in *iPod/iPhone Hardware Specifications*.

If the iPod refuses the `StartIDPS` command by returning a General lingo `ACK` command with a status of 0x04 (Bad Parameter), the accessory must assume it is connected to an iPod that doesn't support the IDPS process. In this case, the accessory must send an `IdentifyDeviceLingoes` command, requesting authentication, within 800 ms.

Table 2-1 (page 48) shows the command traffic for device identification and authentication when using the UART serial port link.

**Note:** Accessory identification and authentication may take place over any of the iPod's transport links—UART, USB, or Bluetooth. If the process fails, the iPod displays a "Accessory not supported" message to the user.

**Table 2-1** Command traffic for UART accessory identification

Step	Action or command	Direction	Comments
1	Wait 80 ms after the iPod turns on Accessory Power (pin 13 in "Hardware Interfaces" in <i>iPod/iPhone Hardware Specifications</i> )	Device	Wait for the iPod's internal bootstrap and wakeup. If the iPod is in Sleep mode, the accessory must repeat Steps 1-5 until the iPod wakes and the accessory receives an <code>ACK</code> command.
2	Send sync byte (0xFF)	Device to iPod	Allow iPod to synchronize to the device's baud rate.

Step	Action or command	Direction	Comments
3	Wait 20 ms	Device	
4	StartIDPS (0x38)	Device to iPod	The accessory sends StartIDPS to start the identification process specified in <a href="#">"Accessory Identification"</a> (page 445).
5	Wait up to 1 second	Device	The device waits for the iPod to send an ACK command acknowledging receipt of StartIDPS.
6	ACK (0x02) of StartIDPS	iPod to Device	The iPod sends a status of 0x00 (OK) to acknowledge receipt of the StartIDPS command.
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 7. Otherwise,</p> <ul style="list-style-type: none"> <li>■ If the accessory receives no ACK command within 1 second, it may retry Step 1 at 1-second intervals as long as iPod Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.</li> <li>■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the iPod does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83). A sample command sequence is listed in <a href="#">Table F-22</a> (page 532).</li> <li>■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.</li> </ul>			
7	SetFIDTokenValues (0x39)	Device to iPod	The accessory sends the iPod a collection of tokens that provide information about it; see <a href="#">"Command 0x39: SetFIDTokenValues"</a> (page 109).
8	RetFIDTokenValueACKs (0x3A)	iPod to Device	The iPod acknowledges receipt of the tokens sent by the SetFIDTokenValues command.
9	EndIDPS (0x3B)	Device to iPod	The accessory terminates the IDPS process, passing an accEndIDPSStatus value of 0x00 to ask the iPod to continue with authentication; see <a href="#">"Command 0x3B: EndIDPS"</a> (page 119).

Step	Action or command	Direction	Comments
10	IDPSStatus (0x3C)	iPod to Device	The iPod acknowledges receipt of EndIDPS and passes a status value of 0x00 to indicate that it has received all required tokens and that authentication will proceed; see <a href="#">"Command 0x3C: IDPSStatus"</a> (page 120).
The iPod or iPhone now initiates the authentication process by sending a <code>GetDevAuthenticationInfo</code> command to the accessory, as shown in <a href="#">Table 2-5</a> (page 55).			

A device must reidentify itself if it receives a ["Command 0x00: RequestIdentify"](#) (page 64) command from the iPod.

Once accessory packet transmission has begun, the maximum time between transmitted data bytes is 25 milliseconds. If the inter-character delay exceeds 25 ms, the iPod discards any packet characters already received, plus any remaining characters received before the start of the next valid packet. The iPod may exceed the 25 ms inter-character timing requirement in its outgoing packets when under heavy system load situations.

One known limitation exists when waking an iPod from Sleep mode: the iPod UART is not available for the first few milliseconds after waking from Sleep. If an external device sends a packet to the iPod while it is asleep, the first packet will be lost. Accessories must follow the steps below to wake an iPod and ensure that the first packet is not lost:

1. Send a sync byte; this should wake the iPod.
2. Wait for 20 ms.
3. Send the command packet with sync byte.

**Note:** Supporting the 3G iPod requires special design considerations. See *"Interfacing With the 3G iPod"* in *iPod/iPhone Hardware Specifications*.

## iAP Signaling and Initialization Using the USB or BT Port Link

When using iAP over USB, initialization involves the USB host detecting the attached iPod and setting its iUI configuration. Upon completion of this process, the host may send General lingo iAP commands to the iPod using the USB HID interface. USB host access to the iAP accessory lingo is enabled only after the host has identified itself and been authenticated by the iPod. Before authentication, only a subset of the General lingo commands are available, as shown in [Table 2-4](#) (page 54).

To use iAP over Bluetooth, the accessory must first establish an RFCOMM protocol session with the iPod. While the session is active, the host may send General lingo iAP commands to the iPod using the RFCOMM channel. Access to the iAP accessory lingo over BT is enabled only after the accessory has identified itself and been authenticated by the iPod. Before authentication, only a subset of the General lingo commands are available, as shown in [Table 2-4](#) (page 54).

To identify an accessory to an iPod or iPhone, using USB or BT, new accessory designs must use the Identify Device Preferences and Settings (IDPS) process described in Appendix B, "[Accessory Identification](#)" (page 445). This process ensures their compatibility with future firmware. A sample command sequence is shown in [Table 2-2](#) (page 51). Existing accessory designs may continue to send the iAP General lingo command 0x13, `IdentifyDeviceLingoes`, identifying the lingoes they support and requesting immediate authentication.

**Table 2-2** Commands for accessory identification via USB or BT

Step	Action or command	Direction	Comments
1	<code>StartIDPS</code> (0x38)	Device to iPod	The accessory sends <code>StartIDPS</code> to start the identification process specified in " <a href="#">Accessory Identification</a> " (page 445).
2	Wait up to 1 second	Device	The device waits for the iPod to send an <code>ACK</code> command acknowledging receipt of <code>StartIDPS</code> .
3	<code>ACK</code> (0x02) of <code>StartIDPS</code>	iPod to Device	The iPod sends a status of 0x00 (OK) to acknowledge receipt of the <code>StartIDPS</code> command.
<p>If the <code>ACK</code> reply to <code>StartIDPS</code> returns a status of 0x00, the accessory proceeds to Step 4. Otherwise,</p> <ul style="list-style-type: none"> <li>■ If the accessory receives no <code>ACK</code> command within 1 second, it may retry Step 1 at 1-second intervals as long as iPod Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.</li> <li>■ If the accessory receives an <code>ACK</code> status of 0x04 (Bad Parameter), the iPod does not support IDPS. Within 800 ms the accessory must send an <code>IdentifyDeviceLingoes</code> command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See "<a href="#">Cancelling a Current Authentication Process With IdentifyDeviceLingoes</a>" (page 83). A sample command sequence is listed in <a href="#">Table F-23</a> (page 534).</li> <li>■ If the accessory receives any other nonzero <code>ACK</code> status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.</li> </ul>			
4	<code>SetFIDTokenValues</code> (0x39)	Device to iPod	The accessory sends the iPod a collection of tokens that provide information about it; see " <a href="#">Command 0x39: SetFIDTokenValues</a> " (page 109).
5	<code>RetFIDTokenValueACKs</code> (0x3A)	iPod to Device	The iPod acknowledges receipt of the tokens sent by the <code>SetFIDTokenValues</code> command.

Step	Action or command	Direction	Comments
6	EndIDPS (0x3B)	Device to iPod	The accessory terminates the IDPS process, passing an <code>accEndIDPSStatus</code> value of 0x00 to ask the iPod to continue with authentication; see "Command 0x3B: EndIDPS" (page 119).
7	IDPSStatus (0x3C)	iPod to Device	The iPod acknowledges receipt of EndIDPS and passes a <code>status</code> value of 0x00 to indicate that it has received all required tokens and that authentication will proceed; see "Command 0x3C: IDPSStatus" (page 120).
The iPod or iPhone now initiates the authentication process by sending a <code>GetDevAuthenticationInfo</code> command to the accessory, as shown in Table 2-5 (page 55).			

In addition to identifying itself at plug-in, a host may need to reidentify during an iAP session if the iPod so requests. If the host receives "Command 0x00: RequestIdentify" (page 64), it must send a `StartIDPS` command and repeat the IDPS and authentication processes. USB hosts do not have to reconfigure iUI when responding to the `RequestIdentify` command from the iPod.

**Note:** An accessory that is newly connected to a sleeping or hibernating iPod cannot expect an immediate response. If it uses wired transport, the accessory should wait for the iPod to turn on Accessory Power. If it uses Bluetooth or is unsuccessful the first time, it should retry the `StartIDPS` command every 1 sec until it gets a response.

## Authentication

Authentication is a mechanism used by the iPod to verify whether an attached device is an authorized accessory and by an accessory to authenticate the iPod, if desired. Certain functionality on the iPod is accessible only after a device has been authenticated as an authorized accessory. This functionality is summarized in Table 2-4 (page 54) and includes the use of any accessory lingo command over the USB or BT transport.

The limited set of General lingo commands listed in Table 2-4 (page 54) as requiring no authentication are free over USB and BT. After the device identification process, authentication may be independently initiated from either the iPod or the device. In the identify and authentication sequences, the accessory should check the ACK status for all commands. If a failure status is returned, the process has failed and should be retried and/or abandoned.



**Note:** To participate in the authentication process, an accessory device must contain an authentication coprocessor provided by Apple.

## Levels of Device Authentication

iAP supports three levels of security in the process by which an iPod may authenticate an accessory device connected to it:

- **None.** Older iPod models through the iPod mini do not support authentication. They operate freely with simple accessories that do not require authentication, but they cannot work with newer or more sophisticated accessories whose range of functionality requires authentication. For details, see “Functional Description” in *iPod/iPhone Hardware Specifications*.
- **Authentication 1.0.** Version 1.0 authentication is based on public and private keys, where each accessory has a private key and every iPod has the associated public key. The accessory authentication process is a part of the iAP General lingo (0x00) command protocol and is controlled by the iPod internal software. Devices identify themselves to the iPod as speaking specific lingoes and supporting authentication. iAP queries the device’s authentication information, sends a challenge to the device, and verifies that the device responds to the challenge correctly.
- **Authentication 2.0.** Version 2.0 authentication is based on standard X.509 Version 3 certificates. Information about this standard can be found at the IETF website: <http://tools.ietf.org/html/rfc3280>. Each certificate is generated and signed by a recognized certificate authority (CA) and has a unique serial number. Authentication 2.0 uses certificate classes to identify the type of accessory being authenticated, as shown in [Table 2-3](#) (page 53).

**IMPORTANT:** Current iPod models support Authentication 1.0, as shown in “[Authentication 1.0 Sample Command Sequence](#)” (page 531), only as a legacy technology, to make them compatible with existing accessory devices. All new accessory designs must use Authentication 2.0.

**Table 2-3** iPod and iPhone authentication coprocessor classes

iPod Class ID	iPhone Class ID	Description
1	4	Automotive use only; enables all authenticated features.
2	5	Nonautomotive use only; enables all authenticated features except the Digital Audio lingo. <b>Deprecated; do not use in new products.</b>
3	6	Nonautomotive use only; enables all authenticated features.

## Authentication Requirements

Because device authentication can take a significant amount of time (up to 75 seconds for Authentication 2.0A), it is performed as a background process. The process starts when the iPod or iPhone sends a `GetDevAuthenticationInfo` command to the device. The device must respond by transmitting its entire `RetDevAuthenticationInfo` command within 1.00 seconds for Authentication 1.0 or 2.00 seconds for Authentication 2.0, including the transmission of all sections of its certificate. When the iPod or iPhone

transmits a `GetDevAuthenticationSignature` command, the device must transmit its `RetDevAuthenticationSignature` response within 7.00 seconds for Authentication 1.0, 65.00 seconds for Authentication 2.0A, or 2.00 seconds for Authentication 2.0B.

All lingo-authenticated commands and features are available to accessories once the iPod has sent the device an `AckDevAuthenticationInfo` command with success status (0x00). This provisional authentication lasts until the iPod sends a `AckDevAuthenticationStatus` command indicating that authentication has finished. The lingo command availability will be revoked if the authentication process fails, the device reidentifies without authentication, the iPod sleeps or powers on, or the device is detached from the iPod. The iPod starts both a timer and a retry counter to guarantee that the authentication process will conclude.

**Table 2-4** Lingo commands requiring authentication

Lingoes	UART port link	USB port link	BT port link
Lingo 0x00: General	No (0x00-0x19, 0x23-0x28, 0x38-0x39, 0x3B) Yes (0x1A-0x1F, 0x29-0x4C <sup>1</sup> )		
Lingo 0x01: Microphone	No (0x00-0x03) Yes (0x04-0x0B)	Yes	N/A
Lingo 0x02: Simple Remote	No (0x00) Yes (0x01-0x04)	Yes	N/A
Lingo 0x03: Display Remote	No (0x00-0x07; 0x1A-0x1E) Yes (0x08-0x19; 0x1F-0x20)	Yes	N/A
Lingo 0x04: Extended Interface	No (0x0000-0x003A) Yes (0x003B-0x0043)	Yes	N/A
Lingo 0x05: Accessory Power	No	Yes	N/A
Lingo 0x06: USB Host Control	Yes	N/A	N/A
Lingo 0x07: RF Tuner	Yes	Yes	N/A
Lingo 0x08: Accessory Equalizer	Yes	Yes	N/A
Lingo 0x09: Sports	Yes	Yes	Yes
Lingo 0x0A: Digital Audio	Yes	Yes	N/A
Lingo 0x0B: Reserved	N/A	N/A	N/A
Lingo 0x0C: Storage	Yes	Yes	Yes
Lingo 0x0D: Reserved	N/A	N/A	N/A
Lingo 0x0E: Location	Yes	Yes	Yes
Lingoes 0x0F-0x1F: Reserved	N/A	N/A	N/A

<sup>1</sup> Preference commands (0x29-0x4C) require authentication on all iPods and iPhones except the 5G iPod; however, getting or setting the line-out preference class (0x03) does not require authentication. Commands 0x42 (DevDataTransfer) and 0x43 (iPodDataTransfer) are not available until authentication has finished.

## iPod Authentication of the Accessory

The process of authenticating the device is initiated by the iPod, based on the settings made in the IDPS process. The authentication options also allow a device to request authentication of an iPod, as described in "Device Authentication of iPod" (page 57).

**IMPORTANT:** The device must send the iPod a `StartIDPS` command (or `Identify` if it must support the 3G iPod) before it tries to send any other iAP commands.

Table 2-5 (page 55) summarizes the authentication process, using Authentication 2.0. This is the authentication process that all new devices should use. Steps 4 and 5 are necessary only if the accessory is unable to use the IDPS process and is initializing using `IdentifyDeviceLingoes`.

**Table 2-5** Command traffic for accessory authentication

Step	Action or command	Direction	Comments
1	<code>GetDevAuthentication-Info (0x14)</code>	iPod to Device	The iPod requests accessory authentication information and starts its timeout timer.
2	<code>RetDevAuthentication-Info (0x15)</code>	Device to iPod	The accessory returns its major and minor authentication version and its X.509 public certificate (see Note 1, below).
3	<code>AckDevAuthentication-Info (0x16)</code>	iPod to Device	The iPod assembles and checks the accessory's X.509 certificate. If it does not support the authentication version, or if the certificate check fails, the iPod sends a value of 0x08 to the accessory (see Note 2, below).
All lingo-authenticated commands are enabled after the authentication version is validated, the lingoes requested by the device are checked against the lingoes allowed by the X.509 certificate, and the certificate has been verified (see Note 3, below).			
4 (non-IDPS only)	<code>GetAccessoryInfo (0x27)</code>	iPod to Device	The iPod queries the accessory for information about it.
5 (non-IDPS only)	<code>RetAccessoryInfo (0x28)</code>	Device to iPod	The accessory returns information about its identity and capabilities.
6	<code>GetDevAuthentication-Signature (0x17)</code>	iPod to Device	The iPod sends a 20-byte random challenge to the accessory and asks it to calculate a digital signature.

Step	Action or command	Direction	Comments
7	RetDevAuthentication-Signature (0x18)	Device to iPod	The accessory returns its digital signature to the iPod within 2 seconds (Authentication 2.0B) or 75 seconds (Authentication 2.0A).
8	AckDevAuthentication-Status (0x19)	iPod to Device	The iPod verifies the signature, using the public key contained in the accessory's X.509 certificate, and returns the status of signature verification.

**Notes:**

1. Not more than 500 certificate bytes may be sent at once. If the X.509 certificate is larger than 500 bytes, the device must divide it into sections, each not larger than 500 bytes, for reassembly by the iPod. The iPod will send a General lingo ACK command for each certificate section up to, but not including, the last one. The accessory must wait for the ACK command before sending the next certificate section in a RetDevAuthenticationInfo packet.
2. The final certificate section is acknowledged by an AckDevAuthenticationInfo command.
3. The iPod parses the X.509 certificate into an allowed lingoes mask. It uses the mask to confirm that the device's identified lingoes are allowed by the certificate. If the device requests lingoes not allowed by the certificate, authentication fails. The iPod also verifies that the certificate is valid.

The iPod and the accessory can perform noncritical operations while background authentication is in progress. The command timeout counter and retry counter are not reset until authentication is complete or has failed.

**Note:** iPod-powered accessories are allowed to draw up to 15 mA current during the Authentication process. This time period begins when the accessory receives a GetDevAuthenticationInfo command from the iPod and ends 500 ms after it receives a successful AckDevAuthenticationStatus command.

Some lingoes requested by a device (such as the General, Accessory Equalizer, and Sports lingoes) cause the iPod to request accessory information after it sends the AckDevAuthenticationInfo command. Accessories should be able to handle both authentication requests and asynchronous information requests from the iPod during the authentication process.

With IDPS all iPod preferences are set before authentication. Accessories that must use IdentifyDeviceLingoes should set any iPod preferences that do not require authentication within 2 seconds of receiving the ACK reply to IdentifyDeviceLingoes, and set iPod preferences that require authentication within 2 seconds of receiving the AckDevAuthenticationInfo command.

**IMPORTANT:** If the device is disconnected during the authentication process, or if authentication does not finish for other reasons, the iPod may refuse to recognize the device for several minutes. With the IDPS process, the first command issued should be `StartIDPS`. This will clear the iPod's state and allow normal identification and authentication to occur. If the iPod does not support IDPS and the accessory's mode of use is such that it might be disconnected and then reconnected during the authentication process, then the device should send the iPod an `IdentifyDeviceLingoes` command with the No Authentication option (see [Table 2-34](#) (page 83)) as its first command every time it is connected. This would cancel any authentication process that might already be running in the iPod from a previous device attachment.

The device may ignore `GetAccessoryInfo` commands from the iPod until it sends a second `IdentifyDeviceLingoes` command with actual lingo information.

If the iPod fails to authenticate the device at any point in the Authentication 1.0 or 2.0 process, and the retry count is exhausted, the iPod may display a "Device Not Supported" message to the user.

## Device Authentication of iPod

The accessory can initiate a process to authenticate the iPod, any time after it has been authenticated by the iPod. This process is summarized in [Table 2-6](#) (page 57). See "[General Lingo Command Summary](#)" (page 60) for information on the individual commands.

**Note:** Device authentication of the iPod is currently supported only for Authentication 2.0.

**Table 2-6** Accessory authentication of the iPod

Step	Action or command	Direction	Comments
1	<code>GetiPodAuthentication-Info (0x1A)</code>	Device to iPod	The accessory requests information to authenticate the iPod.
2	<code>RetiPodAuthentication-Info (0x1B)</code>	iPod to Device	The iPod returns its major and minor authentication version and its X.509 public certificate (see Note, below).
3	<code>AckiPodAuthentication-Info (0x1C)</code>	Device to iPod	The accessory assembles and checks the iPod's X.509 certificate. If the accessory does not support the authentication version, or if the certificate check fails, it returns an error status.
4	<code>GetiPodAuthentication-Signature (0x1D)</code>	Device to iPod	The accessory sends a 20-byte random challenge to the iPod and asks it to calculate a digital signature.
5	<code>RetiPodAuthentication-Signature (0x1E)</code>	iPod to Device	The iPod returns its digital signature to the accessory within 75 seconds. The accessory verifies the signature, using the public key contained in the iPod's X.509 certificate. If verification succeeds, the accessory begins to operate.
6	<code>AckiPodAuthentication-Status (0x1F)</code>	Device to iPod	The accessory returns the status of the digital signature comparison.

**Note:**

1. The iPod's X.509 certificate is sent in a PKCS-7 message containing only the certificate, encoded in DER format. See the IETF document RFC 2311, "S/MIME Version 2 Message Specification" for details about using PKCS-7 messages to transfer X.509 certificates. If necessary, the iPod divides the X.509 certificate into sections, each not larger than the maximum packet size that the accessory specified during the IDPS process, for reassembly by the device. If the accessory has not specified a packet size, the iPod sends 128-byte sections. The accessory must not acknowledge any certificate sections sent by the iPod until the complete certificate has been received and verified.

## Command Packet Formats

This section describes the general format for iAP packets. Use the small packet format for payloads up to 255 bytes. Use the large packet format for payloads greater than 255 bytes.

### Small Packet Format

For command packets whose payloads are 255 bytes or less, use the small packet format. The small packet format is shown in [Table 2-7](#) (page 58).

**Table 2-7** Small packet format

Byte number	Value	Meaning
0x00	0xFF	Sync byte
0x01	0x55	Packet start byte
0x02	0xNN	Packet payload length
0x03	0xNN	Lingo ID
0x04	0xNN	Command ID
0x05...0xNN	0xNN	Command data
(last byte)	0xNN	Packet payload checksum

Note that the command ID and command data format for packets with currently unspecified lingoes may not follow the format indicated here (1 byte command ID, 0xN bytes command data). Also note that a packet payload length of 0x00 is not valid for the small packet format; it is reserved as a marker for the large packet format.

### Large Packet Format

For command packets whose payloads are between 256 bytes and 65535 bytes in length, use the large packet format. The large packet format is shown in [Table 2-8](#) (page 59).

**Table 2-8** Large packet format

Byte number	Value	Meaning
0x00	0xFF	Sync byte
0x01	0x55	Packet start byte
0x02	0x00	Packet payload length marker
0x03	0xNN	Packet payload length (bits 15:8)
0x04	0xNN	Packet payload length (bits 7:0)
0x05	0xNN	Lingo ID
0x06	0xNN	Command ID
0x07...0xNN	0xNN	Command data
(last byte)	0xNN	Packet payload checksum

## Packet Details

The sync byte (0xFF) is not considered part of the packet. It is sent merely to facilitate automatic baud rate detection and correction when using a UART serial port link and, in some cases, to power on the iPod. It is not necessary to send the sync byte when using BT or USB as a link.

The packet payload length is the number of bytes in the packet, not including the sync byte, packet start byte, packet payload length byte, or packet payload checksum byte. That is, it is the length of the command ID, lingo, and command data. Thus, the packet payload data length for a `RequestIdentify` command would be 0x02. The Lingo ID specifies the broad category that the communication falls under. The Command ID is a more specific indication of the significance of the packet and is interpreted differently depending on the Lingo ID.

Unless otherwise specified, the following rules apply:

- All packet data fields larger than 8 bits are sent and received in big-endian format; that is, ordered from the most significant byte to the least significant byte.
- Device command packets that have a valid checksum but contain an invalid parameter, invalid command, or other such failure cause the iPod to respond with an ACK command containing the appropriate error status.
- A packet with an invalid checksum received by iPod is presumed to be invalid and is ignored. No ACK or other command is sent to the device in response to the invalid packet.

**Note:** Unless otherwise specified, all data units larger than bytes must be transferred in a big-endian order; that is, 32 bits should be sent as bits 31:24, followed by bits 23:16, and so forth. Similarly, 16 bits should be sent as bits 15:8, followed by bits 7:0. This includes the 16-bit large packet format payload length: the high byte of the length is sent first, followed by the low byte of the length.

The sum of all the bytes from the packet payload length (or marker, if applicable) to and including the packet payload checksum is 0x00. The checksum must be calculated appropriately, by adding the bytes together as signed 8-bit values, discarding any signed 8-bit overflow, and then negating the sum to create the signed 8-bit checksum byte. All packets received with a nonzero checksum are presumed to be corrupted and will be discarded.

## Lingo 0x00: General Lingo

The General lingo is intended for housekeeping commands and must be supported by all devices. In addition to the General lingo, external devices implement function-specific lingoes. For example, a microphone device attached to the mono microphone input of the 9-pin Audio/Remote connector on the iPod uses the Microphone lingo (0x01). The Simple Remote lingo (0x02) is used by Apple's simple in-line remote control. An external RF transmitter device uses the Accessory Power lingo (0x05).

### General Lingo Command Summary

Table 2-9 (page 60) gives a summary of all commands in the General lingo, including the command ID, the length of the associated data, the first version of the General lingo protocol in which the command is supported, and what device authentication (if any) is required to use the command.

**Table 2-9** General lingo commands

Command	ID	Data length	Protocol version	Authentication for UART transport
RequestIdentify	0x00	0x00	All	None
Identify ( <b>deprecated</b> ; see "General Lingo Command 0x01: Identify" (page 523))	0x01	0x01	All	None
ACK	0x02	0x02 or 0x06	1.00	None
RequestRemoteUIMode	0x03	0x00	1.00	None
ReturnRemoteUIMode	0x04	0x01	1.00	None
EnterRemoteUIMode	0x05	0x00	1.00	None
ExitRemoteUIMode	0x06	0x00	1.00	None
RequestiPodName	0x07	0x00	1.00	None
ReturniPodName	0x08	0xNN	1.00	None



Command	ID	Data length	Protocol version	Authentication for UART transport
RequestiPodSoftwareVersion	0x09	0x00	1.00	None
ReturniPodSoftwareVersion	0x0A	0x03	1.00	None
RequestiPodSerialNum	0x0B	0x00	1.00	None
ReturniPodSerialNum	0x0C	0xNN	1.00	None
RequestiPodModelNum	0x0D	0x00	1.00	None
ReturniPodModelNum	0x0E	0xNN	1.00	None
RequestLingoProtocolVersion	0x0F	0x01	1.00	None
ReturnLingoProtocolVersion	0x10	0x03	1.00	None
Reserved	0x11–0x12	N/A	N/A	N/A
IdentifyDeviceLingoes	0x13	0x0C	1.01	None
GetDevAuthenticationInfo	0x14	0x00	1.01	None
RetDevAuthenticationInfo	0x15	0xNN	1.01	None
AckDevAuthenticationInfo	0x16	0x01	1.01	None
GetDevAuthentication-Signature	0x17	0x11 or 0x15	1.01	None
RetDevAuthentication-Signature	0x18	0xNN	1.01	None
AckDevAuthenticationStatus	0x19	0x01	1.01	None
GetiPodAuthenticationInfo	0x1A	0x00	1.01	Authentication 2.0
RetiPodAuthenticationInfo	0x1B	0xNN	1.01	Authentication 2.0
AckiPodAuthenticationInfo	0x1C	0x01	1.01	Authentication 2.0
GetiPodAuthentication-Signature	0x1D	0xNN	1.01	Authentication 2.0
RetiPodAuthentication-Signature	0x1E	0xNN	1.01	Authentication 2.0
AckiPodAuthenticationStatus	0x1F	0x01	1.01	Authentication 2.0
Reserved	0x20–0x22	N/A	N/A	N/A
NotifyiPodStateChange	0x23	0x01	1.02	None
GetiPodOptions	0x24	0x00	1.05	None

Command	ID	Data length	Protocol version	Authentication for UART transport
RetiPodOptions	0x25	0x08	1.05	None
Reserved	0x26	N/A	N/A	N/A
GetAccessoryInfo	0x27	0xNN	1.04	None
RetAccessoryInfo	0x28	0xNN	1.04	None
GetiPodPreferences	0x29	0x01	1.05	Yes <sup>1</sup>
RetiPodPreferences	0x2A	0x02	1.05	Yes <sup>1</sup>
SetiPodPreferences	0x2B	0x03	1.05	Yes <sup>1</sup>
Reserved	0x2C–0x37	N/A	N/A	N/A
StartIDPS	0x38	0x00	1.09	None
SetFIDTokenValues	0x39	0xNN	1.09	None
RetFIDTokenValueACKs	0x3A	0xNN	1.09	None
EndIDPS	0x3B	0x01	1.09	None
IDPSStatus	0x3C	0x01	1.09	None
Reserved	0x3D–0x3E	N/A	N/A	N/A
OpenDataSessionForProtocol	0x3F	0x05	1.09	Authentication 2.0B
CloseDataSession	0x40	0x04	1.09	Authentication 2.0B
DevACK	0x41	0x04	1.09	Authentication 2.0B
DevDataTransfer	0x42	0xNN	1.09	Authentication 2.0B
iPodDataTransfer	0x43	0xNN	1.09	Authentication 2.0B
Reserved	0x44–0x48	N/A	N/A	N/A
SetEventNotification	0x49	0x0A	1.09	Authentication 2.0B
iPodNotification	0x4A	0xNN	1.09	Authentication 2.0B
GetiPodOptionsForLingo	0x4B	0x01	1.09	Authentication 2.0B
RetiPodOptionsForLingo	0x4C	0x09	1.09	Authentication 2.0B
GetEventNotification	0x4D	0x00	1.09	Authentication 2.0B
RetEventNotification	0x4E	0x08	1.09	Authentication 2.0B

Command	ID	Data length	Protocol version	Authentication for UART transport
GetSupportedEvent-Notification	0x4F	0x00	1.09	Authentication 2.0B
Reserved	0x50	N/A	N/A	N/A
RetSupportedEvent-Notification	0x51	0x08	1.09	Authentication 2.0B
Reserved	0x52–0xFF	N/A	N/A	N/A

<sup>1</sup> Preference commands (0x29–0x2B) require authentication on all iPods and iPhones except the 5G iPod; however, getting or setting the line-out preference class (0x03) does not require authentication.

When first connected to an iPod or iPhone, every accessory must identify itself. New accessory designs must use the Identify Device Preferences and Settings (IDPS) process described in Appendix B, "[Accessory Identification](#)" (page 445). This process ensures their compatibility with future firmware. Existing accessory designs may send the iAP General lingo command 0x13, `IdentifyDeviceLingoes`, identifying the lingoes they support. Accessories that need to support the 3G iPod must use the command described in "[General Lingo Command 0x01: Identify \(Deprecated\)](#)" (page 523). For details, see "[Device Signaling and Initialization](#)" (page 48).

The iPod may send a `RequestIdentify` command to the device to ask it to reidentify itself. There is currently no data defined for this command. Any command returned in response to a `RequestIdentify` packet does not need to have the extra sync bytes and delays used during the device startup process (described in "[Device Signaling and Initialization](#)" (page 48)).

The remaining General lingo commands can be used to obtain general information from the iPod. These commands allow the device to request the name, serial number, model number, and software version number of the iPod. The `RequestLingoProtocolVersion` command allows a device to query the iPod for the lingo protocol versions of all supported lingoes on the iPod. The `ACK` command is used by the iPod to report command error conditions and has an ACK pending feature to notify the requesting device how long to wait for responses to certain commands.

Accessories should send the `GetiPodOptionsForLingo` command to query the iPod for its support of specific features for each lingo. The iPod responds with a `RetiPodOptionsForLingo` command, telling the accessory exactly which options it supports for the specific lingo. With iPods that do not support `GetiPodOptionsForLingo`, an accessory can send `RequestLingoProtocolVersion` to get the lingo version; however, this method forces the accessory to infer features using the method described in "[Protocol Features and Availability](#)" (page 35).

**Note:** Supporting the 3G iPod requires special design considerations. See "Interfacing With the 3G iPod" in *iPod/iPhone Hardware Specifications*.

## History of the General lingo protocol

[Table 2-10](#) (page 64) lists changes introduced with each version of the General lingo protocol.

**Table 2-10** General lingo revision history

Lingo version	Command changes	Features
No version	Add: 0x00, 0x01	Request identification, identify as single lingo device
1.00	Add: 0x02, 0x07–0x10	ACK response, request iPod name, software version, serial number, and model number; support for 38400 and 57600 baud rates
1.01	Add: 0x13–0x1F	Identify as multilingo device, Authentication 1.0 process
1.02	Add: 0x23	Notify device of iPod state changes (Sleep/Power On/Hibernate)
1.03	None	(Internal code restructuring)
1.04	Add: 0x27, 0x28	Get/return accessory device information, Authentication 2.0 process
1.05	Add: 0x24–0x25, 0x29–0x2B	Video browsing preferences
1.06	None	Line-out preferences, two-way Authentication 2.0
1.07	None	Additional video preferences
1.08	None	Different timeout times for <code>GetDevAuthenticationInfo</code> and <code>GetDevAuthenticationSignature</code> .
1.09	Add: 0x38–0x3C, 0x3F–0x43, 0x4A–0x4C	Identify accessories through IDPS, communicate with iPhone OS applications, query iPod for lingo options.

## Command 0x00: RequestIdentify

Direction: iPod to Device

The iPod sends this command to prompt accessories to reidentify themselves. If an accessory receives this command, it must respond with `StartIDPS`, as described in "Device Signaling and Initialization" (page 48). The only exception is for accessories that support the 3G iPod, which must respond with the deprecated "Command 0x01: Identify" (page 523).

**Table 2-11** RequestIdentify packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo. All devices must support this lingo.

Byte number	Value	Comment
4	0x00	Command: RequestIdentify
5	0xFE	Checksum

## Command 0x02: ACK

Direction: iPod to Device

The iPod sends the **ACK** command to notify the device of command completion status and errors. The **ACK** command may come in one of two forms, depending on the status being returned. For any status other than command pending, the normal **ACK** packet structure shown in [Table 2-12](#) (page 65) is used.

**Table 2-12** ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command: ACK
5	0xNN	Command result status. Possible values are shown in <a href="#">Table 2-13</a> (page 65).
6	0xNN	The ID of the command being acknowledged
7	0xNN	Checksum

**Table 2-13** ACK command error codes

Value	Description
0x00	Success (OK)
0x01	ERROR: Unknown database category
0x02	ERROR: Command failed
0x03	ERROR: Out of resources
0x04	ERROR: Bad parameter
0x05	ERROR: Unknown ID
0x06	Command Pending; see <a href="#">Table 2-14</a> (page 66)

Value	Description
0x07	ERROR: Not authenticated
0x08	ERROR: Bad authentication version
0x09	ERROR: Accessory power mode request failed
0x0A	ERROR: Certificate invalid
0x0B	ERROR: Certificate permissions invalid
0x0C	ERROR: File is in use
0x0D	ERROR: Invalid file handle
0x0E	ERROR: Directory not empty
0x0F	ERROR: Operation timed out
0x10	ERROR: Command unavailable in this iPod mode
0x11	ERROR: Invalid accessory resistor ID value
0x12–0x14	Reserved
0x15	ERROR: Maximum number of accessory connections already reached
0x16–0xFF	Reserved

If the status returned by the ACK command is Command Pending, an additional field is added to the ACK packet that represents the amount of time, in milliseconds, that a device should wait to receive the final packet indicating that the current command completed or returned an error status.

After receiving a Command Pending ACK, the device should wait for up to the specified number of milliseconds for a final ACK response. If no final ACK packet is received before the specified amount of time expires, the device should retry the command.

**Table 2-14** ACK packet with Command Pending status

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command: ACK
5	0x06	Command result status: Command Pending
6	0xNN	The ID of the command being acknowledged.

Byte number	Value	Comment
7	0xNN	Maximum amount of time to wait for pending response, in milliseconds (bits 31:24).
8	0xNN	Maximum pending wait, in milliseconds (bits 23:16)
9	0xNN	Maximum pending wait, in milliseconds (bits 15:8)
10	0xNN	Maximum pending wait, in milliseconds (bits 7:0)
11	0xNN	Checksum

## Command 0x03: RequestRemoteUIMode

Direction: Device to iPod

The device requests the Extended Interface mode from the iPod. The iPod responds with "[Command 0x04: ReturnRemoteUIMode](#)" (page 67). This command may be used only if the accessory requests Lingo 0x04 during its identification process.

**Table 2-15** RequestRemoteUIMode packet

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x02	Packet payload length
3	0x00	Lingo ID: General lingo
4	0x03	Command ID: RequestRemoteUIMode
5	0xFB	Packet payload checksum byte

## Command 0x04: ReturnRemoteUIMode

Direction: iPod to Device

The iPod returns the current operating mode of the iPod UI. This is either Standard UI mode or Extended Interface mode. If the returned mode byte is nonzero (true), the iPod is in Extended Interface mode. This command may be used only if the accessory requests Lingo 0x04 during its identification process.

**Table 2-16** ReturnRemoteUIMode packet

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0x03	Packet payload length
3	0x00	Lingo ID: General lingo
4	0x04	Command ID: <code>ReturnRemoteUIMode</code>
5	0xNN	Mode byte. If nonzero (true), the iPod is in Extended Interface Mode. If zero (false), the iPod is in Standard UI mode.
6	0xNN	Packet payload checksum byte

## Command 0x05: `EnterRemoteUIMode`

Direction: Device to iPod

The device sends this command to the iPod to force it to enter the Extended Interface mode. If the iPod is already in the Extended Interface mode, it immediately returns a General lingo `ACK` command packet, notifying the user that the command was successful. This command may be used only if the accessory requests Lingo 0x04 during its identification process.

**Note:** This command fails if the iPod does not detect a valid  $R_{ID}$  resistor. See “Accessory Detect and Identify” in *iPod/iPhone Hardware Specifications*.

If the iPod needs to switch modes, it returns a General lingo `ACK Pending` command packet, informing the device how long it will take the iPod to switch modes. This is followed by an `ACK` packet notifying the device that the iPod successfully changed modes. Devices should honor the timeout returned by the `ACK pending` before assuming the original command has failed.

If audio is playing when the iPod enters Extended Interface mode, the iPod will pause playback. If video is playing, the iPod will stop playback.

**Table 2-17** `EnterRemoteUIMode` packet

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x02	Packet payload length
3	0x00	Lingo ID: General lingo
4	0x05	Command ID: <code>EnterRemoteUIMode</code>
5	0xF9	Packet payload checksum byte



## Command 0x06: ExitRemoteUIMode

Direction: Device to iPod

The device sends this command to the iPod to force it to exit the Extended Interface mode. If the iPod is already in the Standard UI mode, it immediately returns a General lingo ACK command packet, notifying the user that the command was successful. This command may be used only if the accessory requests Lingo 0x04 during its identification process.

If the iPod needs to switch modes, it sends a General lingo ACK Pending command packet informing the device how long it will take the iPod to switch modes. This is followed by an ACK packet notifying the device that the iPod successfully changed modes. Devices should honor the timeout returned by the ACK pending before assuming the original command has failed.

**Table 2-18** ExitRemoteUIMode packet

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x02	Packet payload length
3	0x00	Lingo ID; General lingo
4	0x06	Command ID: ExitRemoteUIMode
5	0xF8	Packet payload checksum byte

## Command 0x07: RequestiPodName

Direction: Device to iPod

Retrieves the name of the iPod. The iPod responds with a ["Command 0x08: ReturniPodName"](#) (page 70) command containing the name of the iPod as a null-terminated UTF-8 character array.

**Table 2-19** RequestiPodName packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x07	Command: RequestiPodName
5	0xF7	Checksum

## Command 0x08: ReturniPodName

Direction: iPod to Device

The iPod sends this command in response to the "[Command 0x07: RequestiPodName](#)" (page 69) message from the device. The iPod name is encoded as a null-terminated UTF-8 character array. If the iPod name has not been modified by the user, it is returned as "iPod".

**Note:** Starting with version 1.02 of the General lingo, the `ReturniPodName` command on Windows-formatted iPods returns the iTunes name of the iPod instead of the Windows volume name.

**Table 2-20** `ReturniPodName` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x08	Command: <code>ReturniPodName</code>
5	0xNN...	The name of the iPod as a null-terminated UTF-8 character array.
(last byte)	0xNN	Checksum

## Command 0x09: RequestiPodSoftwareVersion

Direction: Device to iPod

Retrieves the software version information for the iPod. The iPod responds with a "[Command 0x0A: ReturniPodSoftwareVersion](#)" (page 71) containing the major, minor, and revision version numbers.

**Note:** This command requests the iPod software version and not the version of a particular lingo protocol.

**Table 2-21** `RequestiPodSoftwareVersion` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo

Byte number	Value	Comment
4	0x09	Command: RequestiPodSoftwareVersion
5	0xF5	Checksum

## Command 0x0A: ReturniPodSoftwareVersion

---

Direction: iPod to Device

The iPod sends this command in response to the "[Command 0x09: RequestiPodSoftwareVersion](#)" (page 70) message from the device. The iPod returns each version number as an individual byte, with the major version number sent first. For example, if the major, minor, and revision bytes are returned as 0x01, 0x02, and 0x03, the iPod software version number is 1.02.03.

**Table 2-22** ReturniPodSoftwareVersion packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0A	Command: ReturniPodSoftwareVersion
5	0xNN	iPod major version number.
6	0xNN	iPod minor version number.
7	0xNN	iPod revision version number.
8	0xNN	Checksum

## Command 0x0B: RequestiPodSerialNum

---

Direction: Device to iPod

Retrieves the serial number string of the iPod. The iPod responds with a "[Command 0x0C: ReturniPodSerialNum](#)" (page 72) containing the serial number as a null-terminated UTF-8 character array.

**Table 2-23** RequestiPodSerialNum packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0B	Command: RequestiPodSerialNum
5	0xF3	Checksum

## Command 0x0C: ReturniPodSerialNum

Direction: iPod to Device

The iPod sends this command in response to the "[Command 0x0B: RequestiPodSerialNum](#)" (page 71) message from the device. The iPod serial number is encoded as a null-terminated UTF-8 character array.

**Table 2-24** ReturniPodSerialNum packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0C	Command: ReturniPodSerialNum
5	0xNN...	The iPod serial number, as a null-terminated UTF-8 character array.
(last byte)	0xNN	Checksum

## Command 0x0D: RequestiPodModelNum

Direction: Device to iPod

Retrieves model information for the iPod. The iPod responds with a "[Command 0x0E: ReturniPodModelNum](#)" (page 73) containing the model number of the iPod as a 32-bit integer (model ID) and as a null-terminated UTF-8 character array. If an internal memory error occurs while the iPod is processing this command, the iPod returns an ACK command with the Command Failed error status. The returned model number can be used to determine what iPod hardware has been connected. See "[Command 0x0E: ReturniPodModelNum](#)" (page 73) for details.

**Table 2-25** RequestiPodModelNum packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0D	Command: RequestiPodModelNum
5	0xF1	Checksum

## Command 0x0E: ReturniPodModelNum

Direction: iPod to Device

The iPod sends this command in response to the "[Command 0x0D: RequestiPodModelNum](#)" (page 72) message from the device. The iPod model number is encoded as 32-bit integer (model ID) and as a null-terminated UTF-8 character array.

**WARNING:** Because firmware updates can change an iPod's functionality, you must never use the model number alone to determine its capabilities. Use "[Command 0x0F: RequestLingoProtocolVersion](#)" (page 79) instead.

The iPod model number can be used to identify the size and color of an iPod. Currently, the accessory need only examine the first five characters of the model number to make that determination.

**Table 2-26** ReturniPodModelNum packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0E	Command: ReturniPodModelNum
5	0xNN	iPod Model ID (bits 31:24). See <a href="#">Table 2-27</a> (page 74).
6	0xNN	iPod Model ID (bits 23:16)
7	0xNN	iPod Model ID (bits 15:8)
8	0xNN	iPod Model ID (bits 7:0)
9 ... N	0xNN...	The iPod model number as a null-terminated UTF-8 character array. See <a href="#">Table 2-28</a> (page 75) and <a href="#">Table 2-29</a> (page 76).

Byte number	Value	Comment
(last byte)	0xNN	Checksum

Table 2-27 (page 74) shows the existing iPod model IDs. Table 2-28 (page 75) and Table 2-29 (page 76) list the model number strings returned for each iPod model.

**Table 2-27** iPod model IDs

iPod model ID	iPod hardware
0x0003NNNN	3G iPod. This is the white iPod with 4 buttons above a white click wheel.
0x0004NNNN	iPod mini: original 4 GB model.
0x0005NNNN	4G iPod. This is the white iPod with a gray click wheel.
0x0006NNNN	4G iPod (color display)
0x0007NNNN	2nd generation iPod mini (models M9800 – M9807, 4 GB and 6 GB)
0x000BNNNN	5G iPod
0x000CNNNN	iPod nano
0x0010NNNN	2G iPod nano
0x0011NNNN	iPhone
0x0013NNNN	iPod classic
0x00130100	iPod classic 120 GB
0x0014NNNN	3G iPod nano
0x0015NNNN	iPod touch
0x0017NNNN	4G iPod nano
0x0018NNNN	iPhone 3G
0x0019NNNN	2G touch
0x001BNNNN	iPhone 3GS
0x00130200	iPod classic 160 GB
0x001CNNNN	5G iPod nano
0x001DNNNN	2G touch (2009)

**Table 2-28** iPod model number strings M8948-M9974 and P8948-P9830

iPod model ID strings	iPod hardware
M8948, P8948	3G iPod: 30 GB
M8976, P8976	3G iPod: 10 GB
M9160, P9160	iPod mini: 4 GB silver
M9244, P9244	3G iPod: 20 GB
M9245, P9245	3G iPod: 40 GB
M9268, P9268	4G iPod: 40 GB
M9282, P9282, P9659, P9660, P9661, P9662	4G iPod: 20 GB
M9434, P9434	iPod mini: 4 GB green
M9435, P9435	iPod mini: 4 GB pink
M9436, P9436	iPod mini: 4 GB blue
M9437, P9437	iPod mini: 4 GB gold
M9460, M8946, P8946, P9460	3G iPod: 15 GB
M9575	4G iPod: 20 GB (HP-branded)
M9576	4G iPod: 40 GB (HP-branded)
M9585, P9585	iPod photo: 40 GB
M9586, P9586	iPod photo: 60 GB
M9787	4G iPod: 20 GB black (U2 special edition)
M9800, P9800	2nd Generation (2G) iPod mini: 4 GB silver
M9801, P9801	2G iPod mini: 6 GB silver
M9802, P9802	2G iPod mini: 4 GB blue
M9803, P9803	2G iPod mini: 6 GB blue
M9804, P9804	2G iPod mini: 4 GB pink
M9805, P9805	2G iPod mini: 6 GB pink
M9806, P9806	2G iPod mini: 4 GB green
M9807, P9807	2G iPod mini: 6 GB green
M9829, P9829	2nd generation (2G) iPod photo: 30 GB
M9830, P9830	2G iPod photo: 60 GB

iPod model ID strings	iPod hardware
M9872	2G iPod photo: 30 GB (HP-branded)
M9873	2G iPod photo: 60 GB (HP-branded)
M9973	2G iPod mini: 4 GB silver (HP-branded)
M9974	2G iPod mini: 6 GB silver (HP-branded)

**Table 2-29** iPod model number strings MA002/PA002 and higher

iPod model ID strings	iPod hardware
MA002, PA002, PA148, PA323, MA444, PA444	5G iPod: 30 GB white
MA003, PA003, PA150	5G iPod: 60 GB white
MA004, PA004, PA115	iPod nano: 2 GB white
MA005, PA005, PA116	iPod nano: 4 GB white
MA079, PA079	4G iPod (color display): 20 GB
MA080	4G iPod (color display): 20 GB (HP-branded)
MA099, PA099, PA100	iPod nano: 2 GB black
MA107, PA107, PA108	iPod nano: 4 GB black
MA127	4G iPod (color display): 20 GB black (U2 special edition)
MA146, PA146, PA149, MA446, PA446	5G iPod: 30 GB black
MA147, PA147, PA151	5G iPod: 60 GB black
MA215	4G iPod (color display): 20 GB (Harry Potter special edition)
MA253	5G iPod: 30 GB white (Harry Potter special edition)
MA305	5G iPod: 30 GB black (Harry Potter special edition)
MA350, PA350, PA351	iPod nano: 1 GB white
MA352, PA352, PA353	iPod nano: 1 GB black
MA426, PA426, PA427	2G iPod nano: 4 GB silver
MA428, PA428, PA429	2G iPod nano: 4 GB blue
MA448, PA448, PA449	5G iPod: 80 GB white
MA450, PA450, PA451	5G iPod: 80 GB black
MA452, MA664	5G iPod: 30 GB black (U2 special edition)



iPod model ID strings	iPod hardware
MA477, PA477, PA478	2G iPod nano: 2 GB silver
MA487, PA487, PA488	2G iPod nano: 4 GB green
MA489, PA489, PA490	2G iPod nano: 4 GB pink
MA497, PA497, PA498	2G iPod nano: 8 GB black
MA725, PA725	2G iPod nano: 4 GB red
MA899, PA899	2G iPod nano: 8 GB red
MA501	iPhone: 4 GB
MA712	iPhone: 8 GB
MB384	iPhone: 16 GB
MB046	iPhone 3G: 8 GB black
MB048	iPhone 3G: 16 GB black
MB499	iPhone 3G: 16 GB white
MB029, PB029, PB031	iPod classic: 80 GB silver
MB145, PB145, PB146	iPod classic: 160 GB silver
MB147, PB147, PB148	iPod classic: 80 GB black
MB150, PB150, PB151	iPod classic: 160 GB black
MB562, PB562, PB563, PB864	iPod classic: 120 GB silver
MB565, PB565, PB566	iPod classic: 120 GB black
MA978, PA978, PA979, MB245	3G iPod nano: 4 GB silver
MA980, PA980, PA981, MB247	3G iPod nano: 8 GB silver
MB249, PB249, PB250, MB251	3G iPod nano: 8 GB blue
MB253, PB253, PB254, MB255	3G iPod nano: 8 GB green
MB257, PB257, PB258, MB259	3G iPod nano: 8 GB red
MB261, PB261, PB262, MB263	3G iPod nano: 8 GB black
MB453, PB453, PB454, MB455, MB456	3G iPod nano: 8 GB pink
MA623, PA623, PA624, PA839	1G iPod touch: 8 GB
MA627, PA627, PA628	1G iPod touch: 16 GB

iPod model ID strings	iPod hardware
MB376	1G iPod touch: 32 GB
MB598, MB599	4G iPod nano: 8 GB silver
MB732, PB732, PB733	4G iPod nano: 8 GB blue
MB735, PB735, PB736	4G iPod nano: 8 GB pink
MB739, PB739, PB740	4G iPod nano: 8 GB purple
MB742, PB742, PB743	4G iPod nano: 8 GB orange
MB745, PB745, PB746	4G iPod nano: 8 GB green
MB748, PB748, PB749	4G iPod nano: 8 GB yellow
MB751, MB753, PB751	4G iPod nano: 8 GB red
MB754, PB754, PB755, PB889	4G iPod nano: 8 GB black
MB903, MB904, MB920, PB903, PB904, PB920	4G iPod nano: 16 GB silver
MB905, MB906, MB921, PB905, PB906, PB921	4G iPod nano: 16 GB blue
MB907, MB908, MB922, PB907, PB908, PB922	4G iPod nano: 16 GB pink
MB909, MB910, MB923, PB909, PB910, PB923	4G iPod nano: 16 GB purple
MB911, MB912, MB924, PB911, PB912, PB924	4G iPod nano: 16 GB orange
MB913, MB914, MB925, PB913, PB914, PB925	4G iPod nano: 16 GB green
MB915, MB916, MB926, PB915, PB916, PB926	4G iPod nano: 16 GB yellow
MB917, PB917	4G iPod nano: 16 GB red
MB918, MB919, MB927, PB918, PB919, PB927	4G iPod nano: 16 GB black
MB528, PB528, PB529	2G touch: 8 GB
MB531, PB531, PB532	2G touch: 16 GB
MB533, PB533, PB534	2G touch: 32 GB
MB715, MB735	iPhone 3GS: 16 GB black
MB716, MB736	iPhone 3GS: 16 GB white
MB717, MB737	iPhone 3GS: 32 GB black
MB718, MB738	iPhone 3GS: 32 GB white
MC008	2G touch (2009): 32 GB

iPod model ID strings	iPod hardware
MC011	2G touch (2009): 64 GB
MC027	5G iPod nano: 8 GB silver
MC031	5G iPod nano: 8 GB black
MC034	5G iPod nano: 8 GB purple
MC037	5G iPod nano: 8 GB blue
MC040	5G iPod nano: 8 GB green
MC043	5G iPod nano: 8 GB yellow
MC046	5G iPod nano: 8 GB orange
MC049	5G iPod nano: 8 GB red
MC050	5G iPod nano: 8 GB pink
MC060	5G iPod nano: 16 GB silver
MC062	5G iPod nano: 16 GB black
MC064	5G iPod nano: 16 GB purple
MC066	5G iPod nano: 16 GB blue
MC068	5G iPod nano: 16 GB green
MC070	5G iPod nano: 16 GB yellow
MC072	5G iPod nano: 16 GB orange
MC074	5G iPod nano: 16 GB red
MC075	5G iPod nano: 16 GB pink

## Command 0x0F: RequestLingoProtocolVersion

Direction: Device to iPod

Retrieves version information for any of the lingoes supported by the iPod. The iPod responds with a ["Command 0x10: ReturnLingoProtocolVersion"](#) (page 80) containing the major and minor version information of the requested iPod lingo. This command has one parameter, the lingo whose version information is requested. The iPod returns an ACK command with a bad parameter status if an accessory calls this command with an invalid or unsupported lingo ID. When an iPod does not respond to the `GetiPodOptionsForLingo` command, the accessory may use the `RequestLingoProtocolVersion` command to determine what iAP features are available for each lingo used.

**Table 2-30** RequestLingoProtocolVersion packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x0F	Command: RequestLingoProtocolVersion
5	0xNN	The lingo for which to request version information.
6	0xNN	Checksum

## Command 0x10: ReturnLingoProtocolVersion

---

Direction: iPod to Device

The iPod sends this command in response to the "[Command 0x0F: RequestLingoProtocolVersion](#)" (page 79) message from the device. The major and minor version information for the requested lingo are returned.

**Table 2-31** ReturnLingoProtocolVersion packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x10	Command: ReturnLingoProtocolVersion
5	0xNN	The lingo for which version information is being returned.
6	0xNN	The major protocol version for the given lingo.
7	0xNN	The minor protocol version for the given lingo.
8	0xNN	Checksum

## Command 0x13: IdentifyDeviceLingoes

---

Direction: Device to iPod

**IMPORTANT:** New accessory designs must use the Identify Device Preferences and Settings (IDPS) process described in Appendix B, "[Accessory Identification](#)" (page 445). This process ensures their compatibility with future firmware. Existing accessory designs may continue to send the `IdentifyDeviceLingoes` command, identifying the lingoes they support and requesting immediate authentication.

The device sends this command to signal its presence and to identify its supported lingoes. In response, the iPod sends an `ACK` command. Devices use the `IdentifyDeviceLingoes` command to report all supported lingoes; it must be used in place of the `Identify` (0x01) command.

The `IdentifyDeviceLingoes` command resets all device information set by a previous `Identify` command, including the authentication retry counter and any previously granted authentication access permissions. The payload of this command includes three fields: the Device Lingoes Spoken, Options, and Device ID fields.

**Note:** An accessory attached via the 30-pin connector must always monitor the Accessory Power output from the iPod (pin 13 in "Hardware Interfaces" in *iPod/iPhone Hardware Specifications*). If Accessory Power goes low, even momentarily, the accessory must stop sending iAP commands. After Accessory Power goes high, the accessory must wait at least 80 ms and then send a `StartIDPS` command, following the steps shown in [Table 2-1](#) (page 48) (UART accessories) or [Table 2-2](#) (page 51) (USB or BT accessories).

The `IdentifyDeviceLingoes` command disables all but free lingoes on the current port unless authentication is requested (immediate authentication is also required for Authentication 2.0). For serial ports, this means lingoes 0x00, 0x02, 0x03, and 0x05 may be used, excluding authenticated commands; the USB port will be able to use only the general lingo, 0x00 (see [Table 2-4](#) (page 54)). Devices that register with this command can use only those lingoes that they specifically identify (see [Table 2-33](#) (page 82)).

If any lingo identified by the `IdentifyDeviceLingoes` command can be used by only one device at a time, and that lingo is already in use by a different device, the command will fail but no command failure `ACK` command will be sent to the device. The device can verify that the `IdentifyDeviceLingoes` command has succeeded by sending a free command with valid parameters and checking the iPod's response. The iPod should respond with an `ACK` response with failure status if any lingo is already in use. An identification failure results in the device being able to use only the General lingo (0x00).

This command performs lingo conflict checking to ensure that single-instance lingoes, such as Display Remote, are used on only one port at a time. If the `IdentifyDeviceLingoes` command is acknowledged with a status of 0x15 (Maximum number of accessory connections already reached), the accessory must not send any further iAP traffic until it has been disconnected and reconnected to the iPod.

For sample command sequences in which an accessory identifies its supported lingoes, using `IdentifyDeviceLingoes`, see [Sample Identification Sequences](#) (page 334).

**Table 2-32** `IdentifyDeviceLingoes` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0E	Length of packet payload
3	0x00	Lingo ID: General lingo

Byte number	Value	Comment
4	0x13	Command: IdentifyDeviceLingoes
5-8	0xNN	Device Lingoes Spoken; see <a href="#">Table 2-33</a> (page 82).
9-12	0xNN	Options; see <a href="#">Table 2-34</a> (page 83).
13-16	0xNN	Device ID. Devices must send a unique identifier, supplied by the iPod Authentication Coprocessor, if they require authentication. If a device does not require authentication, it can send the Device ID to 0x00000000 and set the authentication option bits to 0x0.
17	0xNN	Checksum

Use the Device Lingoes Spoken field as a bit field to set each bit corresponding to the lingoes supported by the accessory. For example, if an accessory device supports both the Microphone and Simple Remote lingoes, the bit field is 0x00000007 or the low byte in binary is 00000111.

**Table 2-33** Device Lingoes Spoken bits

Bit	Supported lingo
0	General lingo (must be set by all devices)
1	Microphone lingo
2	Simple Remote lingo
3	Display Remote lingo
4	Extended Interface lingo
5	Accessory Power lingo
6	USB Host Control lingo (deprecated; see " <a href="#">Lingo 0x06: USB Host Control Lingo</a> " (page 525))
7	RF Tuner lingo
8	Accessory Equalizer lingo
9	Sports lingo
10	Digital Audio lingo
11	Reserved; set to 0
12	Storage lingo
31:13	Reserved; set to 0

**Note:** The Location lingo (Lingo 0x0E) is available only if the accessory has identified itself using IDPS. See "Device Signaling and Initialization" (page 48).

The bits of the Options field are defined as shown in Table 2-34 (page 83).

**Table 2-34** IdentifyDeviceLingoes Options bits

Bits	Meaning
1:0	Authentication control bits. These bits have the following meanings: 00 = no authentication is supported or required 01 = defer authentication until an authenticated command is used (Authentication 1.0 only); see Note below 10 = authenticate immediately after identification (required for Authentication 2.0). 11 = reserved
3:2	Power control bits. These bits have the following meanings: 00 = low power only; device requires not more than 5 mA power from the iPod at any time 01 = intermittent high power; device requires more than 5 mA power from iPod (up to 100mA maximum) during playback operation 10 = reserved 11 = constant high power; device requires more than 5 mA power from iPod (up to 100 mA maximum) at all times. This mode must not be declared unless the accessory provides power as specified in "Supplying USB Power" in <i>iPod/iPhone Hardware Specifications</i> .
31:4	Reserved; set to 0

**Note:** Certain lingoes require immediate authentication. Requesting deferred authentication with the Microphone, USB Host Control, RF Tuner, Accessory Equalizer and Digital Audio lingoes results in a command failed ACK return from the iPod.

Devices identifying using the IdentifyDeviceLingoes command receive notifications when the iPod changes state. See "Command 0x23: NotifyiPodStateChange" (page 92) for more details.

**Note:** If a device uses an invalid device ID during an identification attempt, the iPod returns a bad parameter error (0x04) ACK. If the device claims a Device Lingo Spoken that is not supported by the attached iPod, the iPod returns a command failed (0x02) ACK.

## Cancelling a Current Authentication Process With IdentifyDeviceLingoes

For best success calling IdentifyDeviceLingoes, use the following sequence:

1. For UART-based communications, ensure that the Accessory Identify pin (pin 10) is connected to the correct resistors and that the Accessory Detect (pin 20) pin is grounded; see "Accessory Detect and Identify" in *iPod/iPhone Hardware Specifications*. Also, precede all iAP packets by an extra 0xFF autobaud synchronization byte.

2. Send `IdentifyDeviceLingoes` with the lingo mask set to only the General lingo, the option bitmask set to 0x0 (no options), and the device ID set to 0x0. This will cancel any active authentication process on the current iPod accessory port.
3. Wait up to 1 second for the iPod to send an ACK response. After the ACK response, the iPod will send the accessory a `GetAccessoryInfo` command. The accessory must respond with a `RetAccessoryInfo` command, but it must ignore the iPod's acknowledgment of that command.
4. If the iPod responds with an ACK success status within 1 second, proceed to the rest of the device initialization and authentication processes.
5. If the iPod does not respond with an ACK success status within 1 second, repeat Steps 2 and 3 as many times as desired unless the accessory supports the 3G iPod.
6. If the accessory supports the 3G iPod, stop after the third try (3 seconds total elapsed time) and assume that the device is attached to a 3G iPod.

Sample command sequences that illustrate some of these best practices are listed in [Table F-22](#) (page 532) and [Table F-23](#) (page 534).

## Command 0x14: GetDevAuthenticationInfo

---

Direction: iPod to Device

The iPod sends this command to obtain authentication information from the device. The command is sent only if the device has indicated that it supports authentication during its identification process and has passed a valid, nonzero device ID. In response, the device sends "[Command 0x15: RetDevAuthenticationInfo](#)" (page 84).

**Table 2-35**    `GetDevAuthenticationInfo` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x14	Command: <code>GetDevAuthenticationInfo</code>
5	0xEA	Checksum

## Command 0x15: RetDevAuthenticationInfo

---

Direction: Device to iPod



The device indicates the iAP authentication version that it supports by returning this command in response to a "[Command 0x14: GetDevAuthenticationInfo](#)" (page 84) command from the iPod. The authentication version returned by this command must be consistent with the device ID sent during its identification process.

The returned packet may have either of two formats, depending on whether the authentication process is Authentication 1.0 or 2.0. See [Table 2-36](#) (page 85) and [Table 2-37](#) (page 85).

**Table 2-36** RetDevAuthenticationInfo packet, Authentication 1.0

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x15	Command: RetDevAuthenticationInfo
5	0xNN	Authentication protocol major version number
6	0xNN	Authentication protocol minor version number
7	0xNN	Checksum

**Table 2-37** RetDevAuthenticationInfo packet, Authentication 2.0

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x15	Command: RetDevAuthenticationInfo
5	0x02	Authentication major version (0x02)
6	0x00	Authentication minor version (0x00)
7	0xNN	X.509 certificate current section index (0 = first section, 1 = second section, and so on)
8	0xNN	X.509 certificate maximum section index (0 = one section total, 1 = two sections, and so on)
9	0xNN...	X.509 certificate data. If the data length exceeds 500 bytes it must be broken into sections, each not more than 500 bytes.
(last byte)	0xNN	Checksum

**Note:** During Authentication 2.0, not more than 500 certificate bytes may be sent at once. If the X.509 certificate is larger than 500 bytes, the device must divide it into sections, each not larger than 500 bytes, for reassembly by the iPod. The iPod will send an `ACK` command for each certificate section up to, but not including, the last one. The accessory must wait for the `ACK` command before sending the next certificate section. The final certificate section is acknowledged by an `AckDevAuthenticationInfo` command.

## Command 0x16: AckDevAuthenticationInfo

Direction: iPod to Device

The iPod sends this command in response to "[Command 0x15: RetDevAuthenticationInfo](#)" (page 84). It indicates the current state of the device authentication information. If the device receives a nonzero status, the iPod does not support the device's authentication version and the device will fail authentication.

**Note:** The accessory must send certificate data in sections not larger than 500 bytes, for reassembly by the iPod, and must wait for an `ACK` command after each one. The iPod acknowledges the final certificate section with this command.

**Table 2-38** AckDevAuthenticationInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x16	Command: AckDevAuthenticationInfo
5	0xNN	Status of authentication information. Possible values are: 0x00 = Authentication information supported 0x08 = Authentication information unsupported 0x0A = Certificate is invalid 0x0B = Certificate permissions are invalid
6	0xNN	Checksum

## Command 0x17: GetDevAuthenticationSignature

Direction: iPod to Device

The iPod sends this command to authenticate a device that has identified itself as requiring authentication. Authentication occurs either immediately upon identification or when the device attempts to use a restricted lingo or command. The device calculates its digital signature based on the challenge offered by the iPod and sends the results back to the iPod using "[Command 0x18: RetDevAuthenticationSignature](#)" (page 87).

The authentication retry counter is used to track the number of retries. If the returned signature cannot be verified, the iPod responds with a nonzero "[Command 0x19: AckDevAuthenticationStatus](#)" (page 88), followed immediately by another "[Command 0x17: GetDevAuthenticationSignature](#)" (page 86).

The retry counter is set to 0x01 for the first authentication attempt and incremented each time the iPod retries the `GetDevAuthenticationSignature` command. Devices using Authentication 1.0 are allowed up to four retries and a maximum of 30 seconds (up to 7.5 seconds per retry) for the authentication process. Devices using Authentication 2.0 are allowed up to two retries and a maximum of 150 seconds (up to 75 seconds per retry) for the authentication process. If a device fails to respond to the `GetDevAuthenticationSignature` command, the authentication process will fail and the device will not be able to use any of the authenticated commands.

**Table 2-39** `GetDevAuthenticationSignature` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x13 or 0x17	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x17	Command: <code>GetDevAuthenticationSignature</code>
5–20 or 5–24	0xNN...	An offered challenge sent for the device to sign and return (16 bytes for Authentication 1.0, 20 bytes for Authentication 2.0).
NN	0xNN	Authentication retry counter. The authentication process terminates after the maximum retry count or maximum timeout interval has been reached, whichever comes first. When the authentication process fails, only nonauthenticated lingoes and commands are usable.
(last byte)	0xNN	Checksum

## Command 0x18: RetDevAuthenticationSignature

Direction: Device to iPod

The device sends this command to the iPod in response to "[Command 0x17: GetDevAuthenticationSignature](#)" (page 86). The iPod verifies the digital signature, calculated by the device based on the offered challenge. If verification passes, the iPod authenticates the device and updates its lingo and command access permissions accordingly. The authentication status is sent to the device using "[Command 0x19: AckDevAuthenticationStatus](#)" (page 88).

**Table 2-40** RetDevAuthenticationSignature packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x18	Command: RetDevAuthenticationSignature
5	0xNN...	The digital signature calculated by the device, based on the offered challenge (variable length).
(last byte)	0xNN	Checksum

## Command 0x19: AckDevAuthenticationStatus

Direction: iPod to Device

The iPod sends this command to the device in response to the "[Command 0x18: RetDevAuthenticationSignature](#)" (page 87) command. It indicates the current device authentication state. If the device receives a nonzero status, the device has failed authentication and will only be able to use unauthenticated lingo commands.

If the device receives a zero status, the iPod has successfully authenticated the device. The device may then use the requested authenticated lingoes and commands. Optionally, the device may begin the process of authenticating the iPod, by sending "[Command 0x1A: GetiPodAuthenticationInfo](#)" (page 89).

**Table 2-41** AckDevAuthenticationStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x19	Command: AckDevAuthenticationStatus
5	0xNN	Status of the authentication operation: 0x00 = Authentication operation passed All other values = Authentication operation failed; see <a href="#">Table 2-13</a> (page 65)
6	0xNN	Checksum

## Command 0x1A: GetiPodAuthenticationInfo

Direction: Device to iPod

The device sends this command to obtain authentication information from the iPod. The device should send this command only if the device has indicated that it supports authentication during its identification process and the iPod has successfully authenticated the device. (Device authentication is successful when the device receives the "[Command 0x19: AckDevAuthenticationStatus](#)" (page 88) command with a status of 0x00). In response to `GetiPodAuthenticationInfo` the iPod sends "[Command 0x1B: RetiPodAuthenticationInfo](#)" (page 89).

**Table 2-42** `GetiPodAuthenticationInfo` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1A	Command: <code>GetiPodAuthenticationInfo</code>
5	0xE4	Checksum

## Command 0x1B: RetiPodAuthenticationInfo

Direction: iPod to Device

The iPod returns this command in response to "[Command 0x1A: GetiPodAuthenticationInfo](#)" (page 89) from the device.

**Table 2-43** `RetiPodAuthenticationInfo` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1B	Command: <code>RetiPodAuthenticationInfo</code>
5	0x02	Authentication major version (0x02)
6	0x00	Authentication minor version (0x00)

Byte number	Value	Comment
7	0xNN...	X.509 certificate current section index (0 = first section, 1 = second section, and so on)
8	0xNN	X.509 certificate maximum section index (0 = one section total, 1 = two sections, and so on)
9	0xNN	X.509 certificate data. If the data length exceeds 500 bytes it must be broken into sections, each not more than 500 bytes.
(last byte)	0xNN	Checksum

**Note:** Authentication version 2.00 (major version 0x02, minor version 0x00) is the only version supported by iPods that can be authenticated by devices.

## Command 0x1C: AckiPodAuthenticationInfo

Direction: Device to iPod

The device sends this command to the iPod in response to "[Command 0x1B: RetiPodAuthenticationInfo](#)" (page 89). It indicates the current state of the iPod authentication information version. If the device sends a nonzero status, it indicates that it will not be able to authenticate the iPod due to an invalid X.509 certificate.

**Table 2-44** AckiPodAuthenticationInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1C	Command: AckiPodAuthenticationInfo
5	0xNN	Status of the authentication information: 0x00 = authentication information valid Other values = authentication information not valid
6	0xNN	Checksum

## Command 0x1D: GetiPodAuthenticationSignature

Direction: Device to iPod

The device uses this command to send an offered challenge to the iPod for digital signature. In response, the iPod returns its signed challenge to the device using "[Command 0x1E: RetiPodAuthenticationSignature](#)" (page 91). Accessories must implement the authentication retry feature described in "[Command 0x17: GetDevAuthenticationSignature](#)" (page 86), allowing the iPod two retries in 150 seconds. The retry counter must be set to 0x01 in the first `GetiPodAuthenticationSignature` command sent to the iPod and must be incremented for each subsequent attempt. Authentication must fail after either the retry count or maximum response interval have been exceeded since the first `GetiPodAuthenticationSignature` command was sent.

**Table 2-45** `GetiPodAuthenticationSignature` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1D	Command: <code>GetiPodAuthenticationSignature</code>
5–24	0xNN...	An offered challenge for the iPod to sign and return (20 bytes)
25	0x00	Authentication retry counter. A maximum of two retries or 150 seconds, whichever happens first, should occur before the authentication process is terminated.
26	0xNN	Checksum

## Command 0x1E: RetiPodAuthenticationSignature

Direction: iPod to Device

The iPod sends this command to the device in response to "[Command 0x1D: GetiPodAuthenticationSignature](#)" (page 90). The device verifies the digital signature, calculated by the iPod based on the offered challenge, and, if verification passes, authenticates the iPod. The device sends the authentication status to the iPod using "[Command 0x1F: AckiPodAuthenticationStatus](#)" (page 92).

**Table 2-46** `RetiPodAuthenticationSignature` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1E	Command: <code>RetiPodAuthenticationSignature</code>

Byte number	Value	Comment
5 ... <i>N</i>	0x <i>NN</i> ...	The digital signature calculated by the iPod
(last byte)	0x <i>NN</i>	Checksum

## Command 0x1F: AckiPodAuthenticationStatus

---

Direction: Device to iPod

The device sends this command to the iPod in response to "[Command 0x1E: RetiPodAuthenticationSignature](#)" (page 91). It indicates the current iPod authentication state. The device should return a nonzero ACK for each failed authentication attempt.

**Table 2-47** AckiPodAuthenticationStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x1F	Command: AckiPodAuthenticationStatus
5	0x <i>NN</i>	Status of authentication information: 0x00 = authentication operation passed Other values = authentication operation failed
6	0x <i>NN</i>	Checksum

## Command 0x23: NotifyiPodStateChange

---

Direction: iPod to Device

When the iPod power state is about to change, the iPod sends this notification command to devices that identify using IDPS or `IdentifyDeviceLingoes`. If the device identifies using the deprecated command `Identify`, this notification is not sent. The state change byte indicates the specific iPod state transition. If the iPod is switching from a Power On state to a Sleep state, devices must immediately reduce their power consumption below the 5 mA maximum current. When the iPod has transitioned to Hibernate state, self-powered accessories are expected to automatically reidentify themselves 80 ms after accessory power is restored.



**Table 2-48** `NotifyiPodStateChange` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x23	Command: <code>NotifyiPodStateChange</code>
5	0xNN	<p><code>StateChg</code> (1 byte). The iPod state change. Possible values are:</p> <p>0x00 = reserved.</p> <p>0x01 = accessory power going to Hibernate state (no power) and not preserving menu selections or playback context (see “Power States” in <i>iPod/iPhone Hardware Specifications</i>).</p> <p>0x02 = accessory power going to Hibernate state (no power) but preserving menu selections and playback context.</p> <p>0x03 = accessory power going to Sleep state (less than 5 mA current).</p> <p>0x04 = accessory power going to the Power On state.</p> <p>0x05–0xFF = reserved.</p>
6	0xNN	Checksum

**Note:** Firmware version 1.0.0 of the iPod nano reported the iPod state change incorrectly. The `StateChg` byte is decremented by 1, so that 0x00 represents a value of 0x01, 0x01 represents a value of 0x02, and so forth. All later versions of the nano firmware conform to the table above.

## Command 0x24: `GetiPodOptions`

Direction: Device to iPod

**IMPORTANT:** The preferred way to obtain information about the capabilities of an attached iPod is to send “[Command 0x4B: `GetiPodOptionsForLingo`](#)” (page 132). New accessory designs must send `GetiPodOptionsForLingo` first. If and only if an ACK command with Bad Parameter status (0x04) is returned, then the accessory may send `GetiPodOptions` instead.

The accessory device sends this command to ask the iPod to return a 64-bit field that defines the options that the iPod supports. In response, the iPod sends a `RetiPodOptions` command.

**Table 2-49** `GetiPodOptions` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x00	Lingo ID: General lingo
4	0x24	Command ID: <code>GetiPodOptions</code>
5	0xDA	Checksum

## Command 0x25: RetiPodOptions

---

Direction: iPod to Device

The iPod sends this command in response to a `GetiPodOptions` command from the accessory.

**Table 2-50** RetiPodOptions packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0A	Length of packet
3	0x00	Lingo ID: General lingo
4	0x25	Command ID: <code>RetiPodOptions</code>
5	0xNN	Option bits (bits 63:56): Reserved
6	0xNN	Option bits (bits 55:48): Reserved
7	0xNN	Option bits (bits 47:40): Reserved
8	0xNN	Option bits (bits 39:32): Reserved
9	0xNN	Option bits (bits 31:24): Reserved
10	0xNN	Option bits (bits 23:16): Reserved
11	0xNN	Option bits (bits 15:8): Reserved
12	0xNN	Option bits (bits 7:0): Bit 1: iPod supports using <code>SetiPodPreferences</code> to control line-out usage Bit 0: iPod supports video output
13	0xNN	Checksum

## Command 0x27: GetAccessoryInfo

Direction: iPod to Device

The iPod sends this command to devices that identify themselves using the `IdentifyDeviceLingoes` command to obtain certain information from the accessory. The accessory must respond with a `RetAccessoryInfo` command for the required accessory Info Types. The iPod will use the information gathered to:

- Display accessory information in the Settings>About box on the iPod
- Display a message to the user if the iPod firmware needs to be updated to support the accessory
- Display a message to the user if the iPod firmware does not support the accessory

The `GetAccessoryInfo` command sends the Accessory Info Type and the Accessory Info Type parameters to the accessory. [Table 2-52](#) (page 96) lists the number of parameter bytes for the corresponding Accessory Info Types. The iPod requests each of the Accessory Info Types in the order in which they appear in [Table 2-52](#) (page 96). Because the Accessory minimum supported iPod firmware version request (which sends the iPod model number as a parameter) is sent before any Accessory minimum supported lingo version requests, the accessory has the option of changing its responses to those appropriate for the iPod model.

When the `GetAccessoryInfo` command is sent with the accessory minimum supported iPod firmware version info type, the 4-byte iPod model number and the 3-byte iPod firmware version are sent as parameters. See "[Command 0x0E: ReturniPodModelNum](#)" (page 73) for the model number format and "[Command 0x0A: ReturniPodSoftwareVersion](#)" (page 71) for the firmware version format.

When the `GetAccessoryInfo` command is sent with the accessory minimum supported lingo version info type, the 1-byte lingo number for which the iPod is requesting the minimum supported version is sent as a parameter. The iPod will send the `GetAccessoryInfo` command with this Accessory Info Type for every lingo that the accessory indicates it supports.

The iPod begins sending `GetAccessoryInfo` commands as soon as an accessory identifies itself successfully via the `IdentifyDeviceLingoes` command and either enters the background authentication state or does not request authentication. If the accessory does not respond, the iPod waits 5 seconds for a response before timing out and retrying. It retries a maximum of three times.

**Table 2-51** `GetAccessoryInfo` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x27	Command ID: <code>GetAccessoryInfo</code>
5	0xNN	Accessory Info Type (see <a href="#">Table 2-52</a> (page 96))
n...	0xNN...	Accessory Info Type parameters (see <a href="#">Table 2-52</a> (page 96) for length)

Byte number	Value	Comment
(last byte)	0xNN	Checksum

**Table 2-52** Accessory Info Type values

Value	Meaning	Bytes	Requirement
0x00	Accessory info capabilities	0	Required
0x01	Accessory name	0	Required
0x02	Accessory minimum supported iPod firmware version	7	Optional
0x03	Accessory minimum supported lingo version	1	Optional
0x04	Accessory firmware version	0	Required
0x05	Accessory hardware version	0	Required
0x06	Accessory manufacturer	0	Required
0x07	Accessory model number	0	Required
0x08	Accessory serial number	0	Optional
0x09	Accessory incoming maximum payload size	0	Optional
0x0A-0xFF	Reserved		

**Table 2-53** GetAccessoryInfo packet with Accessory Info Type = 0x02

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x0A	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x27	Command ID: GetAccessoryInfo
5	0x02	Accessory Info Type (see <a href="#">Table 2-56</a> (page 98))
6	0xNN	iPod Model ID (bits 31:24)
7	0xNN	iPod Model ID (bits 23:16)
8	0xNN	iPod Model ID (bits 15:8)
9	0xNN	iPod Model ID (bits 7:0)

Byte number	Value	Comment
10	0xNN	iPod firmware major version number
11	0xNN	iPod firmware minor version number
12	0xNN	iPod firmware revision version number
13	0xNN	Checksum

**Table 2-54** GetAccessoryInfo packet with Accessory Info Type = 0x03

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x27	Command ID: GetAccessoryInfo
5	0x03	Accessory Info Type (see <a href="#">Table 2-56</a> (page 98))
6	0xNN	Lingo ID
7	0xNN	Checksum

## Command 0x28: RetAccessoryInfo

Direction: Device to iPod

The accessory device must reply with this command to every receipt of command 0x27, `GetAccessoryInfo`, from the iPod. The data contained in the packet must be responsive to the Accessory Info Type requested by the iPod.

When the `RetAccessoryInfo` command is returning the accessory info capabilities, a bit-field is returned where every set bit represents a supported Accessory Info Type. See [Table 2-51](#) (page 95) for a description of the Accessory Info Type; see [Table 2-56](#) (page 98) for the meanings of its bytes.

**Table 2-55** RetAccessoryInfo packet with Accessory Info Type = 0x00

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x07	Length of packet payload

Byte number	Value	Comment
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0x00	Accessory Info Type. See <a href="#">Table 2-56</a> (page 98)
6	0xNN	Accessory capabilities, bits 31:24; see <a href="#">Table 2-57</a> (page 98))
7	0xNN	Accessory capabilities, bits 23:16
8	0xNN	Accessory capabilities, bits 15:8
9	0xNN	Accessory capabilities, bits 7:0
10	0xNN	Checksum

**Table 2-56** Accessory Info Type values for RetAccessoryInfo

Value	Meaning	Parameter bytes
0x00	Accessory info capabilities	4
0x01	Accessory name	1–64
0x02	Accessory minimum supported iPod firmware version	7
0x03	Accessory minimum supported lingo version	3
0x04	Accessory firmware version	3
0x05	Accessory hardware version	3
0x06	Accessory manufacturer	1–64
0x07	Accessory model number	1–64
0x08	Accessory serial number	1–64
0x09	Accessory incoming maximum payload size	2
0x0A–0xFF	Reserved	N/A

**Table 2-57** Accessory Capabilities bit field

Bit	Capability supported	Accessory requirement
0	Accessory info capabilities	Required; set to 1
1	Accessory name	Required; set to 1
2	Accessory minimum supported iPod firmware version	Optional

Bit	Capability supported	Accessory requirement
3	Accessory minimum supported lingo version	Optional
4	Accessory firmware version	Required; set to 1
5	Accessory hardware version	Required; set to 1
6	Accessory manufacturer	Required; set to 1
7	Accessory model number	Required; set to 1
8	Accessory serial number	Optional
9	Accessory incoming max packet size	Optional
10–31	Reserved	N/A

**IMPORTANT:** Accessories must provide capability information for all required items listed in [Table 2-57](#) (page 98).

The accessory name, manufacturer, model number, and serial number are sent as UTF-8 character arrays and must be less than or equal to 64 bytes (including a null termination character). See [Table 2-57](#) (page 98) for details. Note that even if this condition is met, the iPod may not be capable of displaying all the characters in the array in its About box.

**Table 2-58** RetAccessoryInfo packet with Accessory Info Type = 0x01/0x06/0x07/0x08

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0xNN	Accessory Info Type. See <a href="#">Table 2-56</a> (page 98)
6...	0xNN...	Null-terminated UTF-8 character array
(last byte)	0xNN	Checksum

When the accessory returns the 3-byte minimum supported iPod firmware major/minor/revision version it requires, it also returns the 4-byte iPod model ID. The accessory should therefore store all the model numbers of iPods that support it, along with their corresponding minimum supported iPod firmware versions.

If an unknown or unsupported iPod model ID is sent to the accessory, the accessory should return the RetAccessoryInfo command with the iPod Model ID and 0xFF as the iPod firmware major/minor/revision version numbers.

**Table 2-59** RetAccessoryInfo packet with Accessory Info Type = 0x02

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x0A	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0x02	Accessory Info Type. See <a href="#">Table 2-56</a> (page 98)
6	0xNN	iPod Model ID (bits 31:24)
7	0xNN	iPod Model ID (bits 23:16)
8	0xNN	iPod Model ID (bits 15:8)
9	0xNN	iPod Model ID (bits 7:0)
10	0xNN	minimum supported iPod firmware major version
11	0xNN	minimum supported iPod firmware minor version
12	0xNN	minimum supported iPod firmware revision version
13	0xNN	Checksum

If the accessory's minimum supported iPod firmware version is higher than the iPod firmware version, and one or more of the lingo version numbers is higher than that supported by the iPod, the iPod will display a message to the user indicating that the iPod firmware should be updated.

If the accessory's minimum supported iPod firmware version is smaller than or equal to the iPod firmware version or any of the major/minor/revision numbers are 0xFF, and one or more of the lingo version numbers is higher than that supported by the iPod, the iPod will display a message to the user indicating that the iPod does not support the accessory.

**Table 2-60** RetAccessoryInfo packet with Accessory Info Type = 0x03

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo



Byte number	Value	Comment
5	0x03	Accessory Info Type. See <a href="#">Table 2-56</a> (page 98)
6	0xNN	Lingo ID
7	0xNN	Major protocol version for lingo ID
8	0xNN	Minor protocol version for lingo ID
9	0xNN	Checksum

**Table 2-61** RetAccessoryInfo packet with Accessory Info Type = 0x04/0x05

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0xNN	Accessory Info Type. See <a href="#">Table 2-56</a> (page 98)
6	0xNN	Accessory major version number
7	0xNN	Accessory minor version number
8	0xNN	Accessory revision version number
9	0xNN	Checksum

The accessory's incoming maximum payload size indicates the maximum size of a packet from the iPod that the accessory can support. If the accessory does not return a value, the iPod assumes that the maximum payload size is 1024 bytes. The maximum payload size must be bigger or equal to 128 bytes and smaller or equal to 65536 bytes.

Only iPod packets that contain a UTF-8 character array can be larger than maximum payload size. In that case, the iPod will insert a null termination character into the UTF-8 array to force it to fit into the packet. This does not apply to commands that allow multipacket responses.

**Note:** The maximum payload size restriction does not apply to General lingo packets related to authentication.

**Table 2-62** RetAccessoryInfo packet with Accessory Info Type = 0x09

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)

Byte number	Value	Comment
1	0x55	Start of packet
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x28	Command ID: RetAccessoryInfo
5	0x09	Accessory Info Type. See <a href="#">Table 2-56</a> (page 98)
6	0xNN	Max payload size (bits 15:8)
7	0xNN	Max payload size (bits 7:0)
8	0xNN	Checksum

## Command 0x29: GetiPodPreferences

Direction: Device to iPod

The accessory device sends this command to ask the iPod to return a specific class of preferences set on it, as defined by a preference class ID. In response, the iPod sends a `RetiPodPreferences` command.

**Table 2-63** GetiPodPreferences packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x00	Lingo ID: General lingo
4	0x29	Command ID: GetiPodPreferences
5	0xNN	Preference class ID (see <a href="#">Table 2-64</a> (page 103)). Class IDs 0x00-0x02, 0x08-0x0A, and 0x0C-0x0D are available only if the iPod supports video playback; see " <a href="#">Video Output Preferences</a> " (page 41). Class ID 0x03 is available only if the iPod supports line-out usage.
6	0xNN	Checksum

**Table 2-64** iPod preference class and setting IDs

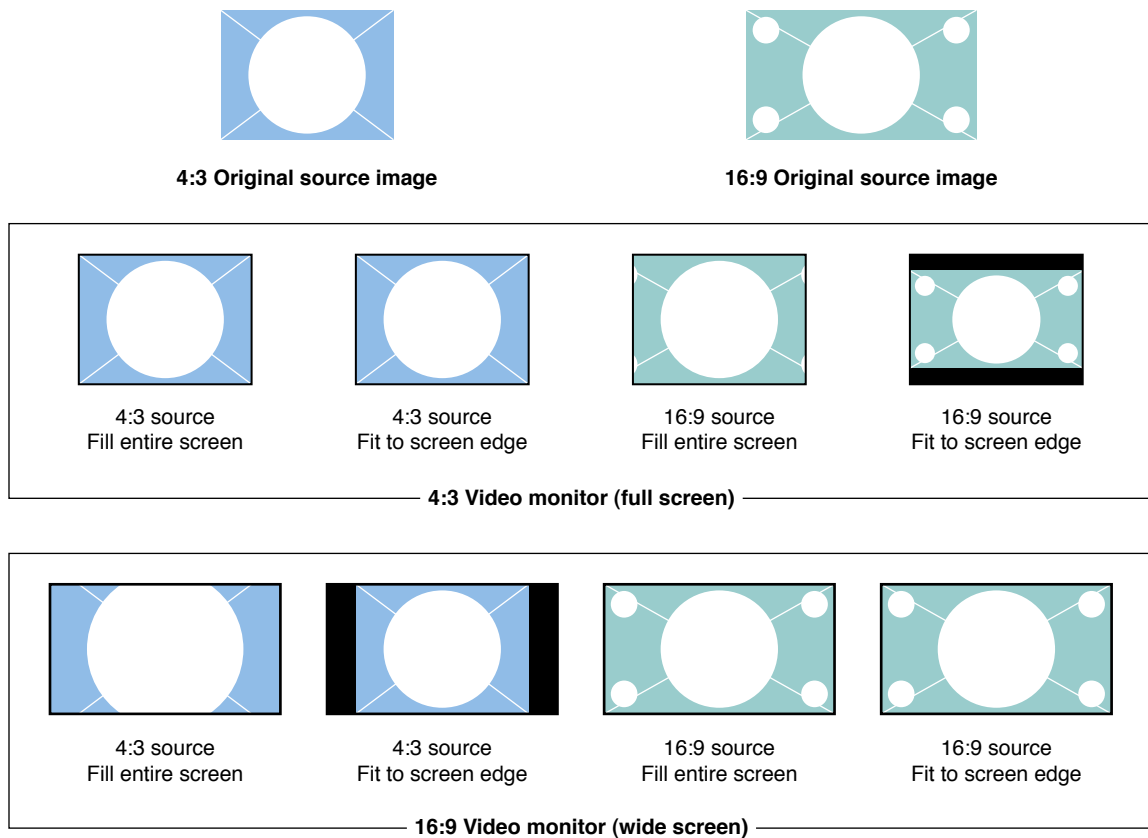
Class ID	Preference	Setting	Setting ID	Description
0x00	Video out setting	Off	0x00	Disables iPod video output of any kind.
		On	0x01	Enables iPod video output based on the other video preferences (NTSC/PAL, screen configuration, etc.).
		Ask user	0x02	If the iPod is not in extended interface mode, makes the UI ask the user to select the video output on/off setting when video playback is started. When the iPod enters extended interface mode, video output mode is automatically enabled. The accessory must not select this setting after entering extended interface mode because the UI will be locked out.
		Reserved	0x03–0xFF	
0x01	Screen configuration	Fill entire screen	0x00	Expand the video image to fill the entire screen without letterbox or pillarbox black bars. The “square” pixel proportions are preserved by enlarging both vertical and horizontal dimensions by the same percentage. Depending on the iPod media source and video monitor destination aspect ratios, this may result in cutting off the top and bottom or left and right edges of the image; see Note 1, below.
		Fit to screen edge	0x01	Expand the video image to screen edge. The iPod enlarges the source media so that its top and bottom or left and right edges end at the screen edge without losing any vertical or horizontal image information. The “square” pixel vertical and horizontal proportions are preserved by being enlarged by the same percentage. Depending on the iPod media source and video monitor destination aspect ratio, this may result in adding either letterbox black bars at the top and bottom of the screen or pillarbox black bars at the left and right of the screen; see Note 2, below.
		Reserved	0x02–0xFF	
0x02	Video signal format	NTSC	0x00	NTSC video format and timing.
		PAL	0x01	PAL video format and timing.
		Reserved	0x02–0xFF	

Class ID	Preference	Setting	Setting ID	Description
0x03	Line-out usage	Not used	0x00	Line-out disabled.
		Used	0x01	Line-out enabled.
		Reserved	0x02-0xFF	
0x04-0x07	Reserved			
0x08	Video-out connection	None	0x00	No connection.
		Composite	0x01	Composite video connection (interlaced video only).
		S-video	0x02	S-video video connection (interlaced video only).
		Component	0x03	Component Y/Pr/Pb video connection (interlaced or progressive scan, depending on the iPod model).
		Reserved	0x04-0xFF	
0x09	Closed captioning	Off	0x00	Closed caption signalling is disabled.
		On	0x01	The iPod overlays closed caption text on the video content before outputting the video signal. The text is not present on line 21, so the video monitor cannot do any closed caption processing. <b>Note:</b> Closed captions and subtitles cannot be enabled at the same time.
		Reserved	0x02-0xFF	
0x0A	Video monitor aspect ratio	0x00	4:3 aspect ratio (fullscreen)	Used when the video output destination monitor has a horizontal to vertical aspect ratio of 4 to 3, such as an original television monitor format. Depending on the media source format and screen configuration preference, the media content may be displayed fullscreen or have letterbox bars. iPods that support widescreen signaling use it to send aspect ratio information to the monitor.
		0x01	16:9 aspect ratio (widescreen)	Used when the video output destination monitor has a horizontal to vertical aspect ratio of 16 to 9, such as the widescreen theater format. Depending on the media source format and screen configuration preference, the media content may be displayed widescreen or have pillarbox bars. iPods that support widescreen signaling use it to send aspect ratio information to the monitor.
		Reserved	0x02-0xFF	

Class ID	Preference	Setting	Setting ID	Description
0x0B	Reserved			
0x0C	Subtitles (see Note 3, below)	0x00	Subtitles off	Subtitle overlays are disabled.
		0x01	Subtitles on	The iPod overlays subtitle text on the video content before outputting the video signal. The video monitor does not need to do any subtitle processing. <b>Note:</b> Closed captions and subtitles cannot be enabled at the same time.
		Reserved	0x02-0xFF	
0x0D	Video alternate audio channel	0x00	Alternate audio off	The alternate audio channel is disabled.
		0x01	Alternate audio on	The alternate audio channel is enabled.
		Reserved	0x02-0xFF	
0x0E-0xFF	Reserved			

**Notes to Table 2-64** (page 103):

1. If the source media has a 4:3 (fullscreen) aspect ratio and the destination monitor has a 16:9 (widescreen) aspect ratio, the video is enlarged to eliminate the pillarbox bars normally present on the left and right of the screen. This may result in the top and bottom edges of the video being enlarged beyond the limits of the screen and cut off. If the source media has a 16:9 (widescreen) aspect ratio and the destination monitor has a 4:3 (fullscreen) aspect ratio, the video is enlarged to eliminate the letterbox bars normally present on the top and bottom of the screen. This may result in the left and right edges of the video being enlarged beyond the limits of the screen and cut off. See [Figure 2-1](#) (page 106).
2. If the source media has a 4:3 (fullscreen) aspect ratio and the destination monitor has a 16:9 (widescreen) aspect ratio, the video is enlarged so that the image fills the screen vertically, but it may have pillarbox black bars on the left and right edges of the screen. If the source media has a 16:9 (widescreen) aspect ratio and the destination monitor has a 4:3 (fullscreen) aspect ratio, the video is enlarged so that the image fills the screen horizontally, but it may have letterbox black bars on the top and bottom edges of the screen. See [Figure 2-1](#) (page 106).
3. On the iPod touch, subtitle display is not a global setting; it is set for each content item.

**Figure 2-1** Screen configuration examples

**Authentication Note:** To receive video signals from the iPhone and from most iPods, and to set their preferences, an accessory must be authenticated; the only class ID that an accessory can get or set without being authenticated is 0x03 (line-out usage). For the best user experience, the iPod should be configured to the device's line out and video out preferences as soon as possible. If a device has not set its preferences, the iPod will make assumptions based on the device's Accessory Identify Resistor and declared lingoes. These assumptions may not match the device's needs, so it is strongly recommended that devices set their iPod preferences promptly. The earliest time at which a device can set its video preferences is immediately after the iPod sends it an `AckDevAuthenticationInfo` command with a value of 0x00 (success status). The 5G iPod is exempt from the authentication requirement.

## Command 0x2A: RetiPodPreferences

Direction: iPod to Device

The iPod sends this command in response to a `GetiPodPreferences` command from the accessory. In byte 5 it echoes the requested preference class ID, and in byte 6 it sends the current preference setting for that class.

**Table 2-65** RetiPodPreferences packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x00	Lingo ID: General lingo
4	0x2A	Command ID: RetiPodPreferences
5	0xNN	Preference class ID (see Table 2-64 (page 103))
6	0xNN	Preference setting ID (see Table 2-64 (page 103))
7	0xNN	Checksum

## Command 0x2B: SetiPodPreferences

Direction: Device to iPod

The accessory device sends this command to the iPod to set a specific preference. It sends the preference class ID in byte 5 and the setting ID in byte 6. If byte 7, restore on exit, is set to 0x01, then the iPod restores the original setting for this preference when the accessory is disconnected; if it is 0x00, it does not perform the restore. Other values of byte 7 are illegal.

**Note:** This command fails if the iPod does not detect a valid  $R_{ID}$  resistor. See “Accessory Detect and Identify” in *iPod/iPhone Hardware Specifications*.

Although iPod options need not be used to turn on video output from the iPod, the iPod has no default settings. It will reflect the options the user has entered in its settings menu, for which the accessory should not make any assumptions. The following are recommended option-setting practices for accessory designs:

- The accessory should set the video signal format (NTSC or PAL) to ensure signal compatibility with the attached display.
- The accessory should set the video monitor aspect ratio, if possible, to ensure image compatibility with the attached display.
- The accessory should select the audio output path (line out or digital audio out) to ensure that audio will be routed appropriately with its associated video.
- The accessory should always select the Restore on Exit flag to return the iPod to its previous state when it is disconnected.

**Note:** Video preferences must be set after the authentication process has begun and after the device has received an `AckDevAuthenticationInfo` command with a value of 0x00. Attempts to set it earlier will result in an ACK command with an error status. See "Authentication Note" (page 106).

**Table 2-66** SetiPodPreferences packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet
3	0x00	Lingo ID: General lingo
4	0x2B	Command ID: SetiPodPreferences
5	0xNN	Preference class ID (see Table 2-64 (page 103))
6	0xNN	Preference setting ID (see Table 2-64 (page 103))
7	0xNN	Restore on exit
8	0xNN	Checksum

**Note:** The iPod line-out state is always restored, regardless of the Restore-on-Exit setting. On some iPods, the video-out state is also always restored, regardless of the Restore-on-Exit setting.

## Command 0x38: StartIDPS

Direction: Device to iPod

This command is sent by the accessory to start the Identify Device Preferences and Settings (IDPS) process. When the iPod sends a General lingo ACK command with a status of Success (0x00) in response to `StartIDPS`, it enters the IDPS process described in "Using IDPS" (page 445). If the accessory sends `StartIDPS` again, while the iPod is in the IDPS process, the iPod will restart the IDPS process.

If the iPod sends a General lingo ACK command with a status of Maximum Connections (0x15) the accessory must not continue with IDPS or any other identification process, because the iPod has already reached its maximum allowed number of accessory connections.

The accessory must start and complete the IDPS process (defined as sending `EndIDPS` successfully) within a total time of 3 seconds after the rising edge of power from the iPod or iPhone on pin 13 (Accessory Power) of the 30-pin connector. The accessory may continue the IDPS process only after it has received an ACK response to `StartIDPS` whose transaction ID matches that of the accessory's most recent `StartIDPS` command. Each `StartIDPS` command renders previous `StartIDPS` commands invalid, even if they are subsequently acknowledged.



After the IDPS process has finished, the iPod sends the accessory an `IDPSStatus` command. If this command indicates an unsuccessful IDPS process, the accessory can retry `StartIDPS` within the 3-second total time limit. If the IDPS process times out, the accessory won't be able to retry IDPS unless it is detached and re-attached, or the iPod sends a `RequestIdentify` command.

If the accessory is in the IDPS process and cannot complete it successfully, it must send `EndIDPS` with a status value of `0x02` before either retrying IDPS or sending `IdentifyDeviceLingoes`. If the accessory does this within 3 seconds of sending `StartIDPS`, the iPod will send an `IDPSStatus` of `0x04` and will let the accessory retry the IDPS process or send `IdentifyDeviceLingoes`. If the accessory fails to do this within 3 seconds of sending `StartIDPS`, the iPod will not let the accessory retry IDPS and will send an `IDPSStatus` of `0x05` in response to the accessory's `EndIDPS` command. After that, the iPod will accept only `IdentifyDeviceLingoes`.

If an accessory has already completed IDPS successfully, inadvertently sending a `StartIDPS` or `IdentifyDeviceLingoes` command resets its authentication and identification states. The accessory must repeat the IDPS and authentication processes.

When an iPod transitions from sleep to power on, it sends a `RequestIdentify` command to the accessory. The accessory must respond with `StartIDPS` and go through the IDPS process. After an iPod transitions from hibernation and the accessory detects accessory power, the accessory must wait at least 80 ms, send `StartIDPS`, and complete the IDPS process.

**Table 2-67** `StartIDPS` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x38	Command ID: <code>StartIDPS</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451).
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	Checksum

## Command 0x39: `SetFIDTokenValues`

Direction: Device to iPod

During the IDPS process, the accessory uses this command to send the iPod a full ID string (FID). This string contains token-value fields that the iPod is able to parse during the accessory identification process. The iPod accepts this command only if the accessory previously initiated the IDPS process by sending a `StartIDPS` command.

The accessory may send this command multiple times with different transaction IDs, but it must put as many token-value fields as possible into each command. No command may exceed a value of 500 bytes in its length-of-packet-payload field. The accessory must wait for the iPod to respond to each `SetFIDTokenValues` command with a `RetFIDTokenValueACKs` command before sending the next. If the same token is sent multiple times, the last value will be stored. If a `SetFIDTokenValues` packet is malformed and the iPod cannot parse one or more token-value fields, the iPod will respond by sending a General lingo `ACK` command with a nonzero status.

**Note:** The accessory must send certain token-value fields to the iPod before the accessory can send `EndIDPS` to complete the IDPS process and proceed with authentication. See [Table 2-70](#) (page 111) for these requirements.

**Table 2-68** SetFIDTokenValues packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x39	Command ID: SetFIDTokenValues
5	0xNN	transID [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451).
6	0xNN	transID [bits 7:0]
7	0xNN	numFIDTokenValues: The number of token-value fields the device is setting by this command. If the number of token-value fields the iPod parses from the command doesn't match this value, the iPod returns a nonzero ACK and accepts no token-value fields.
8-0xNN	0xNN	FIDTokenValues: Token-value fields containing information the device sends to the iPod before going through authentication. See <a href="#">Table 2-69</a> (page 110) for the format of each field.
(last byte)	0xNN	Checksum

**Table 2-69** FIDTokenValues field format

Byte	Size	Name	Description
0	0x01	length	Length of this token-value field in bytes, not including this byte.
1	0x01	InfoByteOne	First byte of token ID; see <a href="#">Table 2-70</a> (page 111).
2	0x01	InfoByteTwo	Second byte of token ID; see <a href="#">Table 2-70</a> (page 111).
3-NN	0xNN	data	Data corresponding to token; see <a href="#">Table 2-71</a> (page 112) through <a href="#">Table 2-80</a> (page 115). All multibyte elements in the data field must be big-endian.

**Table 2-70** FIDTokenValues tokens

infoByteOne	infoByteTwo	Token name	Requirements	Format
0	0	IdentifyToken	Required; must be the first token sent.	See <a href="#">Table 2-71</a> (page 112).
0	1	AccCapsToken	Required	See <a href="#">Table 2-73</a> (page 113).
0	2	AccInfoToken	Accessory Info Types 0x01, 0x04, 0x05, 0x06, and 0x07 required; other types optional. See <a href="#">Table 2-76</a> (page 114).	See <a href="#">Table 2-75</a> (page 113).
0	3	iPodPreferenceToken	Recommended	See <a href="#">Table 2-77</a> (page 114).
0	4	SDKProtocolToken	Required for accessories that communicate with iPhone OS applications.	See <a href="#">Table 2-78</a> (page 114).
0	5	BundleSeedIDPref-Token	Required for accessories that communicate with iPhone OS applications.	See <a href="#">Table 2-79</a> (page 115).
1	0	MicrophoneCapsToken	Required if the accessory includes the Microphone lingo in its IdentifyToken value; see <a href="#">Table 2-71</a> (page 112).	See <a href="#">Table 2-80</a> (page 115).

**Note:** The IDPS process will fail unless the iPod successfully receives all the tokens listed as required in [Table 2-70](#) (page 111). The IdentifyToken token must be sent at the beginning of the first SetFIDTokenValues packet. For the best user experience, the accessory should also return all the recommended tokens that it supports.

The accessory will also fail the IDPS process if it tries to set a preference in the iPodPreferenceToken that requires an accessory capability not previously declared in the AccCapsToken. An example of such a failure would be to declare no line-out support in the AccCapsToken but then try to set a line-out usage preference in the iPodPreferenceToken.

Although an iPodPreferenceToken is not required, iPods do not have default settings and the accessory should make no assumptions about their preferences. During the IDPS process, the accessory should set every preference it will require. The accessory can later change iPod preferences by using the General lingo SetiPodPreferences command.

**Table 2-71** IdentifyToken format

Name	Bytes	Value	Description
length	1	0xNN	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x00	Second byte of token ID.
numLingoes	1	0xNN	Number of bytes in accessoryLingoes.
accessoryLingoes	NN	<var>	One byte for each lingo the accessory supports. For example, 0x00, 0x02, and 0x04 = General lingo + Simple Remote lingo + Extended Interface lingo, with numLingoes set to 0x03 and length set to 0x0C. The General lingo (0x00) must always be included.
DeviceOptions	4	0xNNNNNNNN	Accessory's device options; see Table 2-72 (page 112). The accessory must request immediate Authentication 2.0.
DeviceID	4	0xNNNNNNNN	Accessory's unique identifier, supplied by its iPod Authentication Coprocessor.

**Table 2-72** DeviceOptions bits in IdentifyToken

Bits	Meaning
1:0	Authentication control bits. These bits have the following meanings: 00 = no authentication is supported or required 01 = reserved 10 = authenticate immediately after identification (required for Authentication 2.0). 11 = reserved
3:2	Power control bits. These bits have the following meanings: 00 = low power only; device requires not more than 5 mA power from the iPod at any time 01 = intermittent high power; device requires more than 5 mA power from iPod (up to 100mA maximum) during playback operation 10 = reserved 11 = constant high power; device requires more than 5 mA power from iPod (up to 100 mA maximum) at all times. This mode must not be declared unless the accessory provides power as specified in "Supplying USB Power" in <i>iPod/iPhone Hardware Specifications</i> .
31:4	Reserved; set to 0

**Table 2-73** AccCapsToken format

Name	Bytes	Value	Description
length	1	0x0A	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x01	Second byte of token ID.
accCapsBitmask	8	0xNNNNNNNNNNNNNNNN	Accessory capabilities; see <a href="#">Table 2-74</a> (page 113).

**Table 2-74** Accessory capabilities bit values

Bit	Accessory capability
00	Analog line-out (accessory consumes iPod line-out)
01	Analog line-in (accessory supplies line-in to iPod)
02	Analog video-out (accessory consumes iPod video-out)
03	Reserved (set to 0)
04	USB audio (accessory consumes iPod digital audio)
05-08	Reserved (set to 0)
09	Accessory supports communication with an iPhone OS application.
10	Reserved (set to 0)
11	Accessory checks iPod volume, either by registering for Display Remote lingo notifications or by sending <code>GetiPodStateInfo</code> commands with an <code>infoType</code> of 0x04 or 0x10.
12-63	Reserved (set to 0)

**Table 2-75** AccInfoToken format

Name	Bytes	Value	Description
length	1	0xNN	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x02	Second byte of token ID.
accInfoType	1	0xNN	Accessory info type; see <a href="#">Table 2-76</a> (page 114).
accInfo	NN	<var>	Accessory info; see <a href="#">Table 2-76</a> (page 114).

**Table 2-76** Accessory Info Type values

Value	Meaning	Parameter
0x00	Reserved	
0x01	Accessory name	Null-terminated UTF-8 string (64 bytes maximum, including terminator)
0x02	Reserved	
0x03	Reserved	
0x04	Accessory firmware version	3 bytes
0x05	Accessory hardware version	3 bytes
0x06	Accessory manufacturer	Null-terminated UTF-8 string (64 bytes maximum, including terminator)
0x07	Accessory model number	Null-terminated UTF-8 string (64 bytes maximum, including terminator)
0x08	Accessory serial number	Null-terminated UTF-8 string (64 bytes maximum, including terminator)
0x09	Accessory incoming maximum payload size	2 bytes
0x0A–0xFF	Reserved	

**Table 2-77** iPodPreferenceToken format

Name	Bytes	Value	Description
length	1	0x05	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x03	Second byte of token ID.
iPodPrefClass	1	0xNN	iPod preference class ID; see <a href="#">Table 2-64</a> (page 103).
prefClassSetting	1	0xNN	iPod preference setting ID; see <a href="#">Table 2-64</a> (page 103).
restoreOnExit	1	0x01	Must be set to 0x01.

**Table 2-78** SDKProtocolToken format

Name	Bytes	Value	Description
length	1	0xNN	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.

Name	Bytes	Value	Description
infoByteTwo	1	0x04	Second byte of token ID.
protocolIndex	1	0xNN	A unique protocol index, starting from 1. Subsequent indexes in other <code>SDKProtocolToken</code> token-value fields must increment by 1.
protocolString	NN	<var>	A null-terminated UTF-8 reverse-DNS string (e.g. 'com.acme.gadget'). This string will be compared (without considering case) to strings presented by applications on the iPhone or iPod touch.

**Note:** If an accessory sends a duplicate `protocolString` or repeats a `protocolIndex` number, the last received `SDKProtocolToken` will be used; any information from a previous `SDKProtocolToken` will be overwritten.

**Table 2-79** BundleSeedIDPrefToken format

Name	Bytes	Value	Description
length	1	0x0D	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x05	Second byte of token ID.
BundleSeedIDString	11	<var>	A null-terminated UTF-8 string (e.g. 'A1B2C3D4E5') that identifies the vendor of a preferred application, with case sensitivity. This string is derived from the vendor's App ID assigned by Apple.

**Table 2-80** MicrophoneCapsToken format

Name	Bytes	Value	Description
length	1	0x06	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x01	First byte of token ID.
infoByteTwo	1	0x00	Second byte of token ID.
micCapsBitmask	4	0xNNNNNNNN	Microphone capabilities; see <a href="#">Table 2-81</a> (page 115).

**Table 2-81** Microphone capabilities bits

Bit	Meaning
00	Stereo line input. A value of 0 indicates the device is monophonic only.
01	Stereo or mono line input. This bit should be set only if the microphone supports stereo line input and can switch between stereo and mono modes.

Bit	Meaning
02	Recording level is present and variable.
03	Recording level limit is present.
04	Accessory supports duplex audio; it can play audio output from the iPod while it sends audio input to the iPod.
31:05	Reserved.

## Command 0x3A: RetFIDTokenValueACKs

Direction: iPod to Device

The iPod sends this command in response to each `SetFIDTokenValues` packet from the accessory. The number of token-value fields contained in `FIDTokenValueACKs` matches exactly the number of fields the accessory sent in its `SetFIDTokenValues` command. The different statuses returned to the accessory let it determine whether to retry a failed token-value field.

**Table 2-82** RetFIDTokenValueACKs packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x3A	Command ID: RetFIDTokenValueACKs
5	0xNN	transID [bits 15:8]: Transaction ID of the <code>SetFIDTokenValues</code> packet being responded to; see "Transaction IDs" (page 451).
6	0xNN	transID [bits 7:0]
7	0xNN	numFIDTokenValueACKs: The number of token-value fields the iPod is acknowledging. This number matches the number the accessory sent in the <code>SetFIDTokenValues</code> command.
8-0xNN	0xNN	FIDTokenValueACKs: Acknowledgement values for the token-value fields sent by <code>SetFIDTokenValues</code> . See Table 2-83 (page 117) through Table 2-90 (page 119).
(last byte)	0xNN	Checksum



**Table 2-83** Acknowledgment format for `IdentifyToken`

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x00	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see <a href="#">Table 2-84</a> (page 117).
lingoIDs	0xNN	0xNN	If <code>ACKStatus</code> = 4, IDs of busy lingoes; see <a href="#">Table 2-84</a> (page 117).

**Table 2-84** Acknowledgment status codes

Value	Description
0	Token-value field accepted.
1	Required token-value field failed.
2	Optional token-value field recognized, but failed; the accessory may retry, either with another <code>SetFIDTokenValues</code> command while in the IDPS process or with another iAP command that emulates the desired setting. The accessory should not allow this status return to prevent it from completing IDPS process.
3	Token not supported; the accessory must not retry with this iPod.
4	Lingoes busy. This status responds to an <code>IdentifyToken</code> that tries to register for lingoes that are busy on another port. The ID numbers of the busy lingoes are appended to the token. See <code>lingoIDs</code> in <a href="#">Table 2-83</a> (page 117).
5	Maximum connections reached. This status responds to an <code>IdentifyToken</code> if another accessory is connected and the calling accessory cannot connect to the iPod.
6–255	Reserved

**Table 2-85** Acknowledgment format for `AccCapsToken`

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x01	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see <a href="#">Table 2-84</a> (page 117).

**Table 2-86** Acknowledgment format for `AccInfoToken`

Name	Bytes	Value	Description
length	1	0x04	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x02	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see <a href="#">Table 2-84</a> (page 117).
accInfoType	1	0xNN	Accessory info type in token being acknowledged.

**Table 2-87** Acknowledgment format for `IpodPreferenceToken`

Name	Bytes	Value	Description
length	1	0x04	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x03	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see <a href="#">Table 2-84</a> (page 117).
iPodPrefClass	1	0xNN	iPod preference class in token being acknowledged.

**Table 2-88** Acknowledgment format for `SDKProtocolToken`

Name	Bytes	Value	Description
length	1	0x04	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x04	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see <a href="#">Table 2-84</a> (page 117).
protocolIndex	1	0xNN	Protocol index in token being acknowledged.

**Table 2-89** Acknowledgment format for `BundleSeedIDPrefToken`

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x00	First byte of token ID.
infoByteTwo	1	0x05	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see <a href="#">Table 2-84</a> (page 117).

**Table 2-90** Acknowledgment format for MicCapsToken

Name	Bytes	Value	Description
length	1	0x03	Length of this token-value field in bytes, not including this byte.
infoByteOne	1	0x01	First byte of token ID.
infoByteTwo	1	0x00	Second byte of token ID.
ACKStatus	1	0xNN	Status of acknowledgment; see <a href="#">Table 2-84</a> (page 117).

## Command 0x3B: EndIDPS

Direction: Device to iPod

This command is sent by the accessory to end the IDPS process. The iPod takes different actions depending on the value of `accIDPSStatus`.

If the accessory sends this command while the iPod is not in the IDPS process, the iPod responds with an ACK command that passes a nonzero status. If the iPod is in the IDPS process and the accessory sends this command with an unsupported `accEndIDPSStatus` value, the iPod remains in the IDPS process.

**Table 2-91** EndIDPS packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x3B	Command ID: EndIDPS
5	0xNN	transID [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451).
6	0xNN	transID [bits 7:0]
7	0xNN	accEndIDPSStatus: The accessory's IDPS status, which determines what action the iPod takes after receiving EndIDPS. See <a href="#">Table 2-92</a> (page 119).
8	0xNN	Checksum

**Table 2-92** accEndIDPSStatus values

Value	Actions to be taken
0	The accessory is finished with IDPS, and asks the iPod to continue with authentication if the iPod sends a <code>IDPSStatus</code> command with success status.

Value	Actions to be taken
1	The accessory asks to reset all IDPS information it has sent to the iPod. The accessory must then send another <code>StartIDPS</code> command and successfully complete the IDPS process within the time limit specified in " <a href="#">Command 0x38: StartIDPS</a> " (page 108).
2	The accessory has determined that the iPod doesn't support features the accessory needs. The accessory is abandoning the IDPS process.
3–255	Reserved

**Note:** In the future, an accessory may be connected to an iPod that supports IDPS but doesn't recognize a specific token-value field. To handle this case, the accessory must respond to the `accIDPSStatus` value of 2. For example, if the accessory needs to set a token-value field that the attached iPod cannot parse, it may determine that it can't work with the iPod and hence must discontinue the identification process.

## Command 0x3C: IDPSStatus

Direction: iPod to Device

This command is sent by the iPod after an accessory sends `EndIDPS`. The iPod determines whether all required token-value fields were successfully sent by the accessory, and sends a `status` value that indicates whether authentication will proceed and the iPod will apply the tokens, or the accessory failed to identify itself properly.

**Note:** After receiving an `EndIDPS` with status 0, the iPod evaluates the set of FID tokens that the accessory sent for completeness and consistency. At this point the accessory will fail the IDPS process if it has set a preference in the `iPodPreferenceToken` that requires an accessory capability not declared in the `AccCapsToken`. An example of such an inconsistency would be to declare no line-out support in the `AccCapsToken` but also set a line-out usage preference in the `iPodPreferenceToken`.

**Table 2-93** IDPSStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x3C	Command ID: IDPSStatus
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see " <a href="#">Transaction IDs</a> " (page 451).
6	0xNN	<code>transID</code> [bits 7:0]

Byte number	Value	Comment
7	0xNN	<code>status</code> : A value that indicates whether the accessory successfully identified during the IDPS sequence. Combined with the value of <code>accEndIDPSStatus</code> sent by the accessory, it determines the iPod's action. See <a href="#">Table 2-94</a> (page 121).
8	0xNN	Checksum

**Table 2-94** iPod actions in response to `accEndIDPSStatus` and IDPS status

<code>accEndIDPSStatus</code>	IDPS status	Action taken by the iPod
0	0	The iPod received all required token-value fields; authentication will proceed. Optional token-value fields may or may not have been received, but their status will not block authentication.
	1	One or more required token-value fields were rejected by an <code>FIDTokenValueACK</code> command and were not correctly resent; IDPS fails.
	2	One or more required token-value fields were missing; IDPS fails.
	3	One or more required token-value fields were rejected by an <code>FIDTokenValueACK</code> command and were not correctly resent, plus one or more required token-value fields were missing; IDPS fails.
1	4	The IDPS time limit was not exceeded; the accessory may retry IDPS or send <code>IdentifyDeviceLingoes</code> .
	5	The IDPS time limit was reached; the accessory cannot retry IDPS but may send <code>IdentifyDeviceLingoes</code> .
2	6	The iPod will not accept any token-value fields and will not continue with authentication. The accessory may send <code>IdentifyDeviceLingoes</code> but not <code>StartIDPS</code> .

accEndIDPSStatus	IDPS status	Action taken by the iPod
0	7	<p>IDPS fails for one or more of the following reasons:</p> <ul style="list-style-type: none"> <li>■ The accessory sent an iPod Preference Token for preference class 0x03 (line-out) without indicating line-out support in the <code>accCapsBitmask</code> field of its Accessory Capabilities Token.</li> <li>■ The accessory sent an iPod Preference Token for at least one of preference classes 0x00 (video-out), 0x01 (video-out screen configuration), 0x02 (video-out signal format), 0x08 (video-out connection), or 0x0A (video monitor aspect ratio) without indicating video-out support in the <code>accCapsBitmask</code> field of its Accessory Capabilities Token.</li> <li>■ The accessory set any SDK Protocol Tokens, or a Preferred Bundle Seed ID, without setting bit 09 in its <code>AccCapsToken</code>.</li> <li>■ The accessory identified for the Digital Audio Lingo in its Identify token without setting the USB Audio bit (bit 04) in its <code>AccCapsToken</code>.</li> <li>■ The accessory identified for the Microphone Lingo in its Identify token without setting the Analog Line-In bit (bit 01) in its <code>AccCapsToken</code>.</li> </ul>
NN	8–255	Reserved

**Note:** When a failed IDPS process is retried, the iPod does not retain any successfully set token-value fields from the failed IDPS process. Accessories must send all token-value fields again.

## Command 0x3F: OpenDataSessionForProtocol

Direction: iPod to Device

The iPod sends this command to tell the accessory that a data stream has been opened between the accessory and an iPhone OS application. It sends the accessory the `sessionID` of the data stream and the `protocolIndex` to which the open session corresponds. All communications between the accessory and the application for the given `sessionID` are assumed to use the protocol specified by `protocolIndex`.

The accessory must send a `DevACK` command in response. A success status means the accessory can support a new open data stream. If the accessory sends a `DevACK` command with other than success status, the iPod informs the application that the accessory has refused the data stream connection.

**Table 2-95** OpenDataSessionForProtocol packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet payload

Byte number	Value	Comment
3	0x00	Lingo ID: General lingo
4	0x3F	Command ID: <code>OpenDataSessionForProtocol</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID accessory must use when acknowledging this command; see <a href="#">"Transaction IDs"</a> (page 451).
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>sessionID</code> [bits 15:8]: Session ID for subsequent data transfer commands
8	0xNN	<code>sessionID</code> [bits 7:0]
9	0xNN	<code>protocolIndex</code> : Index of the protocol for which this session is being opened; see <a href="#">Table 2-78</a> (page 114).
10	0xNN	Checksum

## Command 0x40: `CloseDataSession`

Direction: iPod to Device

The iPod sends this command to tell the accessory that the data stream on the given `sessionID` is closing, ending communication. The accessory must send a `DevACK` response, but the data stream will remain closed regardless of the `DevACK` status. If the accessory sends any more `DevDataTransfer` packets for the closed session they will be acknowledged with an Unknown Category status (0x01). The iPod never retries a `CloseDataSession` command.

**Table 2-96** `CloseDataSession` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x40	Command ID: <code>CloseDataSession</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID accessory must use when acknowledging this command; see <a href="#">"Transaction IDs"</a> (page 451).
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>sessionID</code> [bits 15:8]: Session ID of the session being closed
8	0xNN	<code>sessionID</code> [bits 7:0]

Byte number	Value	Comment
9	0xNN	Checksum

## Command 0x41: DevACK

Direction: Device to iPod

The accessory sends this command to acknowledge certain iPod-generated General lingo commands.

**Table 2-97** DevACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x41	Command ID: DevACK
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 451).
6	0xNN	transID [bits 7:0]
7	0xNN	ackStatus: Status of command being acknowledged. See Table 2-13 (page 65).
8	0xNN	cmdID: ID of command being acknowledged.
9	0xNN	Checksum

## Command 0x42: DevDataTransfer

Direction: Device to iPod

The accessory uses this command to send data to an iPhone OS application listening for a specific sessionID, as established by a prior `OpenDataSessionForProtocol` command. The iPod responds with a General lingo ACK command indicating whether or not the application was able to receive the data from this command.

If the accessory sends a `DevDataTransfer` command for a session that is closed or doesn't exist, the iPod acknowledges it with an Unknown Category status (0x01). If the accessory sends a `DevDataTransfer` command for a session that the iPod has opened but which the accessory has not acknowledged successfully, the iPod acknowledges it with a Command Failed status (0x02).

During a communication session, the following rules apply to `DevDataTransfer` commands:

- The accessory should send a `DevDataTransfer` command whenever it has data it wants to stream to an application with an open session.



- The accessory must not send another `DevDataTransfer` packet until the previous one has been acknowledged (as successful or not) or a 500 ms timeout has expired.
- Upon receiving a successful acknowledgment of a `DevDataTransfer` command, the accessory may immediately send a new command containing available data for any session ID.
- If a 500 ms timeout occurs before a `DevDataTransfer` command is acknowledged successfully, the accessory may resend the same packet with the same transaction ID.
- If the accessory receives an `iPodNotification` command for flow control, it must not send any packets (including retries of `DevDataTransfer` packets) during the specified waiting time.

The accessory can choose either small packet format (for packet payload lengths of 255 bytes or less) or large packet format, as defined in ["Command Packet Formats"](#) (page 58). The two `DevDataTransfer` formats are shown in [Table 2-98](#) (page 125) and [Table 2-99](#) (page 125).

**Table 2-98** `DevDataTransfer` small packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x42	Command ID: <code>DevDataTransfer</code>
5	0xNN	transID [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451).
6	0xNN	transID [bits 7:0]
7	0xNN	sessionID [bits 15:8]: Session ID of the connection between the application and the accessory
8	0xNN	sessionID [bits 7:0]
9-NN	0xNN	data: Data the device sends to the listening application.
(last byte)	0xNN	Checksum

**Table 2-99** `DevDataTransfer` large packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x00	Packet payload length marker
3	0xNN	Length of packet payload [bits 15:8]

Byte number	Value	Comment
4	0xNN	Length of packet payload [bits 7:0]
5	0x00	Lingo ID: General lingo
6	0x42	Command ID: DevDataTransfer
7	0xNN	transID [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451).
8	0xNN	transID [bits 7:0]
9	0xNN	sessionID [bits 15:8]: Session ID of the connection between the application and the accessory
10	0xNN	sessionID [bits 7:0]
11-NN	0xNN	data : Data the device sends to the listening application.
(last byte)	0xNN	Checksum

## Command 0x43: iPodDataTransfer

Direction: iPod to Device

The iPod uses this command to send data to an accessory listening for a specific `sessionID`, as established by a prior `OpenDataSessionForProtocol` command. The accessory must respond with a `DevACK` command that passes a command ID of 0x43 and the same transaction ID as contained in the `iPodDataTransfer` packet.

**Note:** The iPod will not send this command until accessory authentication has finished. The Authentication 2.0B process may last up to 2 seconds after accessory identification.

By default, the maximum `iPodDataTransfer` packet payload length is 1,018 bytes. The accessory can specify a different maximum length by sending an `AccInfoToken` with an `accInfoType` of 0x09. The length that can be set ranges from 128 to 65,535 bytes. Specifying less than 128 bytes will revert to 128. The packet payload length is the combined length of the lingo ID, command ID, transaction ID, session ID, and command data; the payload does not include the sync byte, packet start byte, packet payload length bytes, or packet payload checksum byte.

The iPod or iPhone may send data in either small packet format or large packet format, as defined in ["Command Packet Formats"](#) (page 58). The two `iPodDataTransfer` formats are shown in [Table 2-100](#) (page 127) and [Table 2-101](#) (page 127). Accessories must be prepared to receive either format unless they have specified a maximum packet payload length within the limits of the small packet format.

During a communication session, the following rules apply to `iPodDataTransfer` commands:

- The iPod will send an initial `iPodDataTransfer` packet when an application has placed data into the accessory's input queue.
- Upon receiving an acknowledgment of an `iPodDataTransfer` command, the iPod may immediately send a new command containing available data for any session ID.

- The accessory's acknowledgment of an `iPodDataTransfer` command with Success status (0x00) must signify that the packet has been received, it has been moved out of the accessory's lowest-level receive queue, and that the accessory is ready to receive another packet.
- The only circumstance under which an accessory may acknowledge an `iPodDataTransfer` command with a nonzero status is if no prior `OpenDataSessionForProtocol` command has established the command's session ID. In this case it must send a Bad Parameter (0x04) status, and the iPod may close the session.
- If an `iPodDataTransfer` command is acknowledged with an error status, the iPod may optionally purge its outbound packet queue of all pending `iPodDataTransfer` packets for that session ID. The iPod will not otherwise discard any data in a session.
- If an accessory's data receive queue cannot accept another packet, then the accessory must not acknowledge the `iPodDataTransfer` command until packet acceptance is restored. This may cause the iPod to retry the `iPodDataTransfer` command up to ten times. If the iPod receives no acknowledgment after the tenth retry, it may close the session.
- If a 500 ms timeout occurs before an `iPodDataTransfer` command is acknowledged, the iPod will resend the same packet with the same transaction ID.

Table 2-100 `iPodDataTransfer` small packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x43	Command ID: <code>iPodDataTransfer</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see "Transaction IDs" (page 451).
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>sessionID</code> [bits 15:8]: Session ID of the connection between the application and the accessory
8	0xNN	<code>sessionID</code> [bits 7:0]
9-NN	0xNN	<code>data</code> : Data the application sends to the listening accessory.
(last byte)	0xNN	Checksum

Table 2-101 `iPodDataTransfer` large packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x00	Packet payload length marker
3	0xNN	Length of packet payload [bits 15:8]
4	0xNN	Length of packet payload [bits 7:0]
5	0x00	Lingo ID: General lingo
6	0x43	Command ID: <code>iPodDataTransfer</code>
7	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451).
8	0xNN	<code>transID</code> [bits 7:0]
9	0xNN	<code>sessionID</code> [bits 15:8]: Session ID of the connection between the application and the accessory
10	0xNN	<code>sessionID</code> [bits 7:0]
11-NN	0xNN	<code>data</code> : Data the application sends to the listening accessory.
(last byte)	0xNN	Checksum

## Command 0x49: SetEventNotification

Direction: Device to iPod

This command enables asynchronous remote event notifications for specific iPod events, as described in [iPod Event Notifications](#) (page 449).

When the accessory identifies itself, it can check the attached iPod's notification support by sending a `GetSupportedEventNotification` command. If the iPod supports notifications, it returns a `RetSupportedEventNotification` command confirming that it supports at least Flow Control notifications (bit 2 in the 64-bit big-endian notification bitmask). If the accessory needs to send notifications of types that the attached iPod supports, it must then send this command to register for one or more of the notifications listed in [Table 2-103](#) (page 129).

The iPod acknowledges this command with a General Lingo `ACK` command reporting Status OK (0x00).

If the accessory registers for camera notifications, the iPod also sends an `iPodNotification` command reporting its current Camera mode. The iPod may send these commands in any order.

For a suggested sequence of commands to enable iPod notifications, see [iPod Event Notifications](#) (page 449).

**Table 2-102** `SetEventNotification` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x0C	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x49	Command ID: <code>SetEventNotification</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command. If the iPod does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	<code>transID</code> [bits 7:0]
7	0x00	Event notification mask (bits 63:56)
8	0x00	Event notification mask (bits 55:48)
9	0x00	Event notification mask (bits 47:40)
10	0x00	Event notification mask (bits 39:32)
11	0x00	Event notification mask (bits 31:24)
12	0x00	Event notification mask (bits 23:16)
13	0x00	Event notification mask (bits 15:8)
14	0xNN	Event notification mask (bits 7:0); at least bit 2 must be set (see <a href="#">Table 2-103</a> (page 129))
15	0xNN	Checksum

**Table 2-103** Notification bitmask bits

Bit number	Description	Notes
0-1	Reserved; set to 0	
2	Flow control	Must always set to 1. If and only if the accessory uses IDPS, Flow Control notifications are enabled by default. See " <a href="#">Accessory Identification</a> " (page 445).
3	Radio tagging status	See " <a href="#">iTunes Tagging</a> " (page 463).
4	Camera status	See " <a href="#">Accessory Control of the iPod 5G nano Camera</a> " (page 161).
5-63	Reserved; set to 0	

## Command 0x4A: iPodNotification

Direction: iPod to Device

If an accessory has registered for notifications, using `SetEventNotification`, the iPod sends this command to the accessory every time there is a change in the state of any of the processes for which notification was requested. The notification timings and payloads are different for each of the notification types listed in [Table 2-103](#) (page 129):

- If Flow Control notifications are enabled, the iPod sends a Flow Control notification whenever the incoming queue is full and the iPod is unable to accept more packets. When an accessory receives this notification, it must stop sending packets to the iPod/iPhone for the wait time specified by the notification packet shown in [Table 2-104](#) (page 130). If transaction IDs are being used, the Flow Control notification also returns the transaction ID of the packet that caused the overflow. This packet was not added to the incoming data queue; hence the accessory can use the overflow transaction ID to determine which packets need to be resent after the wait time.
- If the accessory registers for Radio Tagging notifications, the iPod sends notifications when tagging information is available or when a tagging operation is initiated (either from the iPod's user interface or from the accessory via the Simple Remote lingo `RadioButtonStatus` command). The notification packet for Radio Tagging is shown in [Table 2-105](#) (page 131).
- If the accessory registers for Camera notifications, the iPod sends a notification immediately after receiving `SetEventNotification` and subsequently whenever its Camera mode changes. It passes the state of its Camera mode using the packet format shown in [Table 2-107](#) (page 132). Because the iPod's notification mechanism is asynchronous, the accessory may receive this packet before its `SetEventNotification` command has been acknowledged. The iPod may also send notifications to the accessory at any time as the result of changes in the Camera application's state that may be unrelated to any action taken by the accessory. For full details see "[Accessory Control of the iPod 5G nano Camera](#)" (page 161).

**Note:** The accessory must not acknowledge `iPodNotification` commands and must simply ignore any that it cannot handle. The iPod does not retry these commands.

**Table 2-104** `iPodNotification` packet for Flow Control Notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0B or 0x07	Length of packet payload: 0x0B if the iPod supports IDPS, 0x07 otherwise.
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: <code>iPodNotification</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command. If the iPod does not support IDPS, this and the next byte are omitted and the value of byte 2 is 0x07.
6	0xNN	<code>transID</code> [bits 7:0]
7	0x02	Notification type: Flow Control
8	0xNN	Wait time in ms (bits 31:24)
9	0xNN	Wait time in ms (bits 23:16)

Byte number	Value	Comment
10	0xNN	Wait time in ms (bits 15:8)
11	0xNN	Wait time in ms (bits 7:0)
12	0xNN	Overflow transaction ID (bits 15:8); Transaction ID of the packet that caused the iPod or iPhone's incoming data queue to overflow. If the iPod does not support IDPS, this and the next byte are omitted.
13	0xNN	Overflow transaction ID (bits 7:0)
14	0xNN	Checksum

**Table 2-105** `iPodNotification` packet for Radio Tagging Notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x09	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: <code>iPodNotification</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command. If the iPod does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	<code>transID</code> [bits 7:0]
7	0x03	Notification type: Radio Tagging
8	0xNN	Tag status; see <a href="#">Table 2-106</a> (page 131).
9	0xNN	Checksum

**Table 2-106** `iPodNotification` payload for Radio Tagging notifications

Byte value	Description
0x00	Tagging operation successful
0x01	Tagging operation failed
0x02	Information available for tagging (all required RT+ information has been received)
0x03	Information not available for tagging (tagging is not possible because the RT+ information has been reset due to a song change, station change, loss of signal, etc.)
0x04-0xFF	Reserved

**Table 2-107** `iPodNotification` packet for Camera notifications

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4A	Command ID: <code>iPodNotification</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command. If the iPod does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	<code>transID</code> [bits 7:0]
7	0x04	Notification type: Camera Notifications
8	0xNN	Payload; see <a href="#">Table 2-108</a> (page 132).
9	0xNN	Checksum

**Table 2-108** `iPodNotification` payload for Camera Notifications

Byte value	Description
0x00	Camera App Off
0x01-0x02	Reserved
0x03	Preview
0x04	Recording
0x05-0xFF	Reserved

## Command 0x4B: `GetiPodOptionsForLingo`

Direction: Device to iPod

The accessory sends this command to ask the iPod to return a 64-bit field that defines the options that the iPod supports for a specific lingo, identified in `LingoID`. In response, the iPod sends a `RetiPodOptionsForLingo` command.

If the device requests options for the General lingo (0x00) and the iPod returns an ACK with nonzero status, then the iPod does not support `GetiPodOptionsForLingo`; the accessory should use `RequestLingoProtocolVersion` instead. If the device requests options for any other lingo and the iPod returns a nonzero ACK, that lingo is not listed in [Table 2-110](#) (page 133) or is not supported by the iPod on the port being used; no `RetiPodOptionsForLingo` command will be returned.



**Table 2-109** GetiPodOptionsForLingo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x00	Lingo ID: General lingo
4	0x4B	Command ID: GetiPodOptionsForLingo
5	0xNN	LingoID: ID of lingo for which options are requested; see <a href="#">Table 2-110</a> (page 133).
6	0xNN	Checksum

**Table 2-110** RetiPodOptionsForLingo option bits

Lingo	Lingo ID	Bit	Description
General	0x00	0x00	Line out usage
		0x01	Video output
		0x02	NTSC video signal format
		0x03	PAL video signal format
		0x04	Composite video out connection
		0x05	S-Video video out connection
		0x06	Component video out connection
		0x07	Closed Captioning (video)
		0x08	Video aspect ratio 4:3 (fullscreen)
		0x09	Video aspect ratio 6:9 (widescreen)
		0x0A	Subtitles (video)
		0x0B	Video Alternate Audio Channel
		0x0C	Reserved
		0x0D	Communication with iPhone OS 3.x applications
		0x0E	iPod notifications
		0x0F-0x3F	Reserved
Microphone	0x01	0x00-0x3F	Reserved

Lingo	Lingo ID	Bit	Description
Simple Remote	0x02	0x00	Context-specific controls
		0x01	Audio media controls
		0x02	Video media controls
		0x03	Image media controls
		0x04	Sports media controls
		0x05-0x3F	Reserved
Display Remote	0x03	0x00	Volume control
		0x01	Absolute Volume control
		0x02-0x3F	Reserved
Extended Interface	0x04	0x00	Video browsing
		0x01	Extended Interface enhancements
		0x02	Nested playlists
		0x03	Reserved
		0x04	Supports Set Display Image
		0x05-0x3F	Reserved
Accessory Power	0x05	0x00-0x3F	Reserved
RF Tuner	0x07	0x00	RDS Raw Mode support
		0x01	HD Radio Tuning support
		0x02	AM Radio Tuning support
		0x03-0x3F	Reserved
Accessory Equalizer	0x08	0x00-0x3F	Reserved
Sports	0x09	0x00	Reserved
		0x01	Nike + iPod Cardio Equipment
		0x02-0x3F	Reserved
Digital Audio	0x0A	0x00	A/V Synchronization
		0x01-0x3F	Reserved
Storage	0x0C	0x00	iTunes Tagging

Lingo	Lingo ID	Bit	Description
		0x01	Nike + iPod Cardio Equipment
		0x02-0x3F	Reserved
Location	0x0E	0x00	iPod accepts NMEA GPS location data
		0x01	iPod can send location assistance data
		0x02-0x3F	Reserved

[Table 2-111](#) (page 135) lists a sample command flow that queries an iPod's options for most of the iAP lingoies.

**Table 2-111** Sample GetiPodOptionsForLingo and RetiPodOptionsForLingo commands

Step	Accessory command	iPod command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> <li>■ If the accessory receives no ACK command within 1 second, it may retry Step 1 at 1-second intervals as long as iPod Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.</li> <li>■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the iPod does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoies command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See "<a href="#">Cancelling a Current Authentication Process With IdentifyDeviceLingoies</a>" (page 83).</li> <li>■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.</li> </ul>			
3	GetiPodOptions-ForLingo		getting options for General Lingo

Step	Accessory command	iPod command	Comment
4		RetiPodOptions-ForLingo	returning options of 0000000000006FFF (Line out usage   Video output   NTSC video signal format   PAL video signal format   Composite video out connection   S-Video video out connection   Component video out connection   Closed captioning (video)   Video aspect ratio 4:3 (fullscreen)   Video aspect ratio 6:9 (widescreen)   Subtitles (video)   Video Alternate Audio Channel   App communication capable   iPod notifications) for General Lingo
5	GetiPodOptions-ForLingo		getting options for Microphone Lingo
6		RetiPodOptions-ForLingo	returning options of 0000000000000000 for Microphone Lingo
7	GetiPodOptions-ForLingo		getting options for Simple Remote Lingo
8		RetiPodOptions-ForLingo	returning options of 000000000000000F (Audio media controls   Video media controls   Image media controls   context-specific controls) for Simple Remote Lingo
9	GetiPodOptions-ForLingo		getting options for Display Remote Lingo
10		RetiPodOptions-ForLingo	returning options of 0000000000000001 (Volume control) for Display Remote Lingo
11	GetiPodOptions-ForLingo		getting options for Extended Interface Lingo
12		RetiPodOptions-ForLingo	returning options of 0000000000000001 (Video browsing) for Extended Interface Lingo
13	GetiPodOptions-ForLingo		getting options for Accessory Power Lingo
14		RetiPodOptions-ForLingo	returning options of 0000000000000000 (none) for Accessory Power Lingo

Step	Accessory command	iPod command	Comment
15	GetiPodOptions-ForLingo		getting options for Accessory Equalizer Lingo
16		RetiPodOptions-ForLingo	returning options of 0000000000000000 (none) for Accessory Equalizer Lingo
17	GetiPodOptions-ForLingo		getting options for Sports Lingo
18		RetiPodOptions-ForLingo	returning options of 0000000000000002 (Nike + iPod cardio equipment) for Sports Lingo
19	GetiPodOptions-ForLingo		getting options for Digital Audio Lingo
20		RetiPodOptions-ForLingo	returning options of 0000000000000000 (none) for Digital Audio Lingo
21	GetiPodOptions-ForLingo		getting options for Storage Lingo
22		RetiPodOptions-ForLingo	returning options of 0000000000000003 (iTunes tagging   Nike + iPod cardio equipment) for Storage Lingo
23	GetiPodOptions-ForLingo		getting options for Location Lingo
24		RetiPodOptions-ForLingo	returning options of 0000000000000003 (iPod accepts NMEA GPS location data   iPod can send location assistance data) for Location Lingo
25	SetFIDTokenValues		setting 8 FID tokens ; AccInfoToken = Acc name (iPod Accessory Name); AccInfoToken = Acc FW version (v1.2.1); AccInfoToken = Acc HW version (v1.0.0); AccInfoToken = Acc manufacturer (Apple Inc.); AccInfoToken = Acc model number (MA1390LL/A); SDKProtocolToken = 1 (com.apple.protocolMain); SDKProtocolToken = 2 (com.apple.protocolAlternative); BundleSeedIDString = 24D4XFAF43

Step	Accessory command	iPod command	Comment
26		RetFIDTokenValueACKs	8 ACKs for FID tokens ; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; SDKProtocolToken = (1) accepted; SDKProtocolToken = (2) accepted; BundleSeedIDPrefToken = accepted
27	SetFIDTokenValues		setting 1 FID tokens ; IdentifyToken = (lingoes: 0/2   options 0x00000006   device ID 0x00000200)
28		RetFIDTokenValueACKs	1 ACKs for FID tokens ; IdentifyToken = accepted
29	SetFIDTokenValues		setting 1 FID tokens ; AccCapsToken = 0x00000000000000A17
30		RetFIDTokenValueACKs	1 ACKs for FID tokens ; AccCapsToken = accepted
31	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'selected')
32		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (line out usage) accepted
33	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'ask' for preference class 'video out setting' with restore on exit 'selected')
34		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video out setting) accepted
35	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'widescreen' for preference class 'screen configuration' with restore on exit 'selected')
36		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (screen configuration) accepted

Step	Accessory command	iPod command	Comment
37	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'PAL' for preference class 'video format setting' with restore on exit 'selected')
38		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video format setting) accepted
39	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'component' for preference class 'video connection' with restore on exit 'selected')
40		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video connection) accepted
41	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting '16:9' for preference class 'aspect ratio' with restore on exit 'selected')
42		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (aspect ratio) accepted
43	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'component' for preference class 'video connection' with restore on exit 'selected')
44		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video connection) accepted
45	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'on' for preference class 'video subtitles' with restore on exit 'selected')
46		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (video subtitles) accepted
47	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'on' for preference class 'alternate audio channel' with restore on exit 'selected')

Step	Accessory command	iPod command	Comment
48		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (alternate audio channel) accepted
49	SetFIDTokenValues		setting 1 FID tokens ; iPodPreferenceToken = (setting 'on' for preference class 'closed captions' with restore on exit 'selected')
50		RetFIDTokenValueACKs	1 ACKs for FID tokens ; iPodPreferenceToken = (closed captions) accepted
51	EndIDPS		status 'finished with IDPS; proceed to authentication'
52		IDPSStatus	status 'ready for auth'
53		GetDevAuthentication-Info	no params
54	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1; cert data: ...
55		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
56	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1; cert data: ...
57		AckDevAuthentication-Info	acknowledging 'auth info supported'
58		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
59	RetDevAuthentication-Signature		returning signature '...'
60		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'

## Command 0x4C: RetiPodOptionsForLingo

Direction: iPod to Device

The iPod sends this command in response to a `GetiPodOptionsForLingo` command from the accessory. It returns the ID of the lingo for which options were requested in `LingoID` and the option bits as a 64-bit big-endian field in bytes 6-13. [Table 2-110](#) (page 133) lists the available option bit values for various lingoos, and [Table 2-111](#) (page 135) lists a sample command flow that queries an iPod's options for those lingoos.



**Table 2-112** RetiPodOptionsForLingo packet

Byte number	Value	Comment	
0	0xFF	Sync byte (required only for UART serial)	
1	0x55	Start of packet (SOP)	
2	0x0B	Length of packet	
3	0x00	Lingo ID: General lingo	
4	0x4C	Command ID: RetiPodOptionsForLingo	
5	0xNN	LingoID: ID of lingo for which options were requested.	
6	0xNN	Option bits (bits 63:56)	See <a href="#">Table 2-110</a> (page 133)
7	0xNN	Option bits (bits 55:48)	
8	0xNN	Option bits (bits 47:40)	
9	0xNN	Option bits (bits 39:32)	
10	0xNN	Option bits (bits 31:24)	
11	0xNN	Option bits (bits 23:16)	
12	0xNN	Option bits (bits 15:8)	
13	0xNN	Option bits (bits 7:0)	
14	0xNN	Checksum	

## Command 0x4D: GetEventNotification

Direction: Device to iPod

The accessory sends this command to the iPod to ask for the iPod's current 64-bit big-endian notification bitmask.

**Table 2-113** GetEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4D	Command ID: GetEventNotification

Byte number	Value	Comment
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the iPod does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0xNN	Checksum

## Command 0x4E: RetEventNotification

Direction: iPod to Device

The iPod sends this command to the accessory in response to a `GetEventNotification` command. It passes its current 64-bit big-endian notification bitmask.

**Table 2-114** RetEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0C	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4E	Command ID: RetEventNotification
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the iPod does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0x00	Event notification mask (bits 63:56)
8	0x00	Event notification mask (bits 55:48)
9	0x00	Event notification mask (bits 47:40)
10	0x00	Event notification mask (bits 39:32)
11	0x00	Event notification mask (bits 31:24)
12	0x00	Event notification mask (bits 23:16)
13	0x00	Event notification mask (bits 15:8)
14	0xNN	Event notification mask (bits 7:0); see <a href="#">Table 2-103</a> (page 129).
15	0xNN	Checksum

## Command 0x4F: GetSupportedEventNotification

Direction: Device to iPod

The accessory sends this command to the iPod to query the iPod's support for notifications. The iPod responds with a `RetSupportedEventNotification` command.

**Table 2-115** GetSupportedEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x4F	Command ID: <code>GetSupportedEventNotification</code>
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the iPod does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]
7	0xNN	Checksum

## Command 0x51: RetSupportedEventNotification

Direction: iPod to Device

The iPod sends this command to the accessory in response to a `GetSupportedEventNotification` command. It passes a 64-bit big-endian bitmask that defines the types of notifications that the iPod supports.

**Table 2-116** RetSupportedEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0C	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x51	Command ID: <code>RetSupportedEventNotification</code>
5	0xNN	transID [bits 15:8]: Transaction ID of this command. If the iPod does not support IDPS, omit this and the next byte and subtract 2 from the value of byte 2.
6	0xNN	transID [bits 7:0]

Byte number	Value	Comment
7	0x00	Event notification mask (bits 63:56)
8	0x00	Event notification mask (bits 55:48)
9	0x00	Event notification mask (bits 47:40)
10	0x00	Event notification mask (bits 39:32)
11	0x00	Event notification mask (bits 31:24)
12	0x00	Event notification mask (bits 23:16)
13	0x00	Event notification mask (bits 15:8)
14	0xNN	Event notification mask (bits 7:0); see <a href="#">Table 2-103</a> (page 129).
15	0xNN	Checksum

# Accessory Lingo

The iPod Accessory Protocol defines a number of different accessory lingoes. This chapter describes these lingoes and their commands.

[Table 3-1](#) (page 145) shows the available accessory lingoes and their IDs.

**Table 3-1** iPod accessory lingoes

Lingo	ID	Notes
Microphone	0x01	
Simple Remote	0x02	
Display Remote	0x03	
Extended Interface	0x04	See <a href="#">The Extended Interface Protocol</a> (page 341). For a sample command sequence that declares the Extended Interface lingo, see <a href="#">Table 3-271</a> (page 334).
Accessory Power	0x05	
USB Host Control	0x06	Deprecated; see " <a href="#">Lingo 0x06: USB Host Control Lingo</a> " (page 525).
RF Tuner	0x07	
Accessory Equalizer	0x08	
Sports	0x09	
Digital Audio	0x0A	
Reserved	0x0B	
Storage	0x0C	
Reserved	0x0D	
Location	0x0E	
Reserved	0x0F–0xFF	

## Command Timings

Some iAP commands sent by the iPod take a significant time to execute, and some of these retry if the device does not acknowledge the first attempt. Timeout information and the number of times the command tries to execute are shown in [Table 3-2](#) (page 146).

**Table 3-2** Select iPod command timings

Lingo	Command	Timeout	Tries
Microphone (0x01)	GetDevAck (0x05)	200 ms	1
	GetDevCaps (0x07)	200 ms	1
	GetDevCtrl (0x09)	200 ms	1
	SetDevCtrl (0x0B)	200 ms	1
RF Tuner (0x07)	GetCaps (0x01)	200 ms	1
	GetCtrl (0x03)	200 ms	1
	SetCtrl (0x05)	200 ms	1
	GetBand (0x06)	200 ms	1
	SetBand (0x08)	200 ms	1
	GetFreq (0x09)	200 ms	1
	SetFreq (0x0B)	200 ms	1
	GetMode (0x0C)	200 ms	1
	SetMode (0x0E)	200 ms	1
	GetSeekRssi (0x0F)	200 ms	1
	SetSeekRssi (0x11)	200 ms	1
	SeekStart (0x12)	200 ms	1
	GetStatus (0x14)	200 ms	1
	GetStatusNotifyMask (0x16)	200 ms	1
	SetStatusNotifyMask (0x18)	200 ms	1
	GetRdsReadyStatus (0x1A)	200 ms	1
	GetRdsData (0x1C)	200 ms	1
	GetRdsNotifyMask (0x1E)	200 ms	1

Lingo	Command	Timeout	Tries
	SetRdsNotifyMask (0x20)	200 ms	1
Accessory Equalizer (0x08)	GetCurrentEQIndex (0x01)	2500 ms	3
	SetCurrentEQIndex (0x03)	3000 ms	3
	GetEQSettingCount (0x04)	3000 ms	3
	GetEQIndexName (0x06)	3000 ms	3
Sports (0x09)	GetDeviceCaps (0x03)	500 ms	2
Digital Audio (0x0A)	GetAccSampleRateCaps (0x02)	3000 ms	3
Location (0x0E)	GetDevCaps (0x01)	500 ms	1
	GetDevControl (0x03)	500 ms	1
	SetDevControl (0x05)	500 ms	1
	GetDevData (0x06)	500 ms	1
	SetDevData (0x08)	500 ms	1

## Lingo 0x01: Microphone Lingo

The Microphone lingo enables combination microphone and speaker accessory devices to record and play back audio. Initial microphone devices supported one input mode (mono) and one sample rate (8 kHz). The increased iPod mass storage disk capacities enable the option of supporting a stereo input mode and higher audio sample rates. With these changes, iPods may be used for high-quality mobile audio recording.

**Note:** The 3G iPod, iPod mini, and first generation iPod nano do not support the microphone lingo; it is not possible to create a voice recorder for these products. The 5G iPod, 2G nano, iPod classic, and 3G nano support both stereo and monophonic microphones through the 30-pin connector. The 4G iPod only supports monophonic microphones plugged into the 9-pin audio/remote connector; it cannot record audio from a microphone connected to the 30-pin connector. See “The 9-Pin Audio/Remote Connector” in the chapter “Historical Information” in *iPod/iPhone Hardware Specifications*.

The Microphone lingo is defined such that the iPod initiates commands and the accessory device responds to these commands; that is, the iPod sends commands to the device and the device responds with data or ACK commands.

When the iPod detects a device speaking the Microphone lingo, it may transition into a recorder application where it can create and manage recordings. Based on the microphone device capabilities, the iPod recording application may choose to change its appearance based on the presence or absence of certain microphone features. The device should indicate its capabilities to the iPod on request. These capabilities may include:

- Stereo line input source
- Stereo/mono control
- Recording level control
- Recording level limiter

**Note:** Accessories using Authentication 2.0 and identifying for the Microphone lingo may have audio routed to them by default, based on their resistor ID, identification method and/or iPod model. If a microphone accessory does not support line-out (i.e. it does not have a built-in speaker) and it uses IDPS for identification, its `AccCapsToken` must not confirm line-out support (see [Table 2-73](#) (page 113)).

Microphone accessory devices can draw power from the iPod or supply power to the iPod. Accessory device power management is important as iPods transition to a smaller physical size at the same time as trying to extend battery life. An accessory using the Microphone lingo must remain in low-power mode by default. It is allowed to transition to high-power mode only if it receives an `iPodModeChange` command with a `Mode` value of `0x00` or `0x02` (begin audio recording or playback), as listed in [Table 3-9](#) (page 152), and also only if it has declared the intermittent high-power option during its identification process. The accessory must return to low-power mode within 100 ms of receiving an `iPodModeChange` command with a `Mode` value of `0x01` or `0x03` (end audio recording or playback). Like all accessories, an accessory using the Microphone lingo must comply with the power requirements of the iPod's Sleep and Hibernate states, as specified in ["iPod Power States and Accessory Power"](#) (page 341). Accessories are informed of iPod state changes by receiving a General lingo `NotifyiPodStateChange` command.

The microphone device is responsible for keeping its power consumption below the maximum allowed limits for each iPod state; see ["Accessory Power Policy"](#) (page 39). Accessory power is completely shut off when the iPod enters the Hibernate state. On reset or power up, the accessory device must be in low power state (consuming less than 5 mA) with the amplifier off (that is, with audio input and output disabled).

Microphone state information must be retained locally by the device while uninterrupted accessory power (either high or low power) is available. If accessory power is turned off, device state information may be lost. Devices are not expected to retain state information across accessory power down cycles (Hibernate mode).

iPod playback volume level changes may require the device to support Display Remote lingo (`0x03`) functionality.

[Table 3-3](#) (page 149) lists the commands available as part of the Microphone lingo.



**Note:** Legacy Microphone lingo commands 0x00–0x03 are disabled for devices using the 30-pin connector. They are superseded by "Command 0x06: iPodModeChange" (page 151), which returns an ACK. Deprecated commands 0x00–0x03 are documented in "9-Pin Audio/Remote Connector Commands" (page 520).

**Table 3-3** Microphone lingo command summary

Command	ID	Data length	Protocol Version	Connector	Authentication Required
BeginRecord	0x00	0x00	All	9-pin Audio/Remote	No
EndRecord	0x01	0x00	All	9-pin Audio/Remote	No
BeginPlayback	0x02	0x00	All	9-pin Audio/Remote	No
EndPlayback	0x03	0x00	All	9-pin Audio/Remote	No
ACK	0x04	0x02	1.01	30-pin	Yes
GetDevAck	0x05	0x00	1.01	30-pin	Yes
iPodModeChange	0x06	0x01	1.01	30-pin	Yes
GetDevCaps	0x07	0x00	1.01	30-pin	Yes
RetDevCaps	0x08	0x04	1.01	30-pin	Yes
GetDevCtrl	0x09	0x00	1.01	30-pin	Yes
RetDevCtrl	0x0A	0x02	1.01	30-pin	Yes
SetDevCtrl	0x0B	0x02	1.01	30-pin	Yes
Reserved	0x0C–0xFF	N/A	N/A	N/A	N/A

## Command History of the Microphone Lingo

Table 3-4 (page 149) shows the history of command changes in the Microphone lingo.

**Table 3-4** Microphone lingo command history

Lingo version	Command changes	Features
No version	Add: 0x00–0x03	Microphone begin/end record/playback notification commands, 9-pin Audio/Remote connector only
1.00	None	Version number available through <code>RequestLingoProtocol - Version</code>
1.01	Add: 0x04–0x0B	ACK, mode change, capabilities, and control support (30-pin connector only)

The commands described in the following sections are used only with the 30-pin connector. For information about 9-pin connector commands, see ["9-Pin Audio/Remote Connector Commands"](#) (page 520).

**Note:** Devices must return responses to iPod commands in the order in which the commands were received and within the specified time limits. Failing to send responses in the order commands were received or exceeding the command timeout limits could cause a communications failure and result in the device being considered not present by the iPod.

## Command 0x04: ACK

Direction: Device to iPod

The microphone device sends this command in response to a command sent from the iPod. Note that the commands 0x00–0x03 do not require an ACK response. The device sends an ACK response when a command that does not return any data has completed, a bad parameter is received, or an unsupported or invalid command is received.

**Table 3-5** ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x04	Command ID: ACK
5	0xNN	The command result status. See <a href="#">Table 3-6</a> (page 150) for the possible values.
6	0xNN	The ID of the command for which the response is being sent.
7	0xNN	Checksum

[Table 3-6](#) (page 150) shows the possible values of the command result status byte.

**Table 3-6** Command result values

Byte	Meaning
0x00	Success (OK)
0x01	Not applicable: the Microphone lingo does not use this error value.
0x02	ERROR: Command failed. Sent in response to a valid command if that command did not succeed.
0x03	ERROR: Out of resources. This indicates that an iPod internal allocation failed.

Byte	Meaning
0x04	ERROR: Bad parameter. The command or input parameters are invalid.
0x05	Not applicable: the Microphone lingo does not use this error value.
0x06	Reserved
0x07	ERROR: The device is not authenticated to use this lingo command.
0x08	ERROR: Mismatched authentication protocol version.
0x09 - 0xFF	Reserved

## Command 0x05: GetDevAck

Direction: iPod to Device

The iPod sends this command to get an **ACK** response from a microphone device. The iPod uses this command to “ping” the device and determine that it is present and ready to accept commands. In response, the device sends the **ACK** command with command status OK.

The timeout for this command is 200 ms (0.2 second). The device must respond within the allotted time; the iPod will not retry the command. If the device does not respond within the specified time, the command will fail and the device may be considered not present by the iPod.

**Table 3-7** GetDevAck packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x05	Command ID: GetDevAck
5	0xF8	Checksum

## Command 0x06: iPodModeChange

Direction: iPod to Device

The iPod sends this command to the microphone device when an audio recording or playback event occurs. The microphone device uses the `iPodModeChange` command to configure its inputs or outputs and power consumption level for the specified mode. In response, the device sends the **ACK** command with the command status OK. The device sends the **ACK** command when the device has completed its mode change.

The iPod does not wait to receive an ACK command from the device in response to the mode change. It may continue sending other commands to the device after it has sent the mode change command.

**Table 3-8** iPodModeChange packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x06	Command ID: iPodModeChange
5	0xNN	Mode. See <a href="#">Table 3-9</a> (page 152).
6	0xNN	Checksum

[Table 3-9](#) (page 152) lists the possible values of the Mode byte.

**Table 3-9** Mode values

Value	Meaning
0x00	<p>Begin audio recording mode.</p> <p>When it receives this command, the device can activate its microphone recording inputs or outputs connected to the iPod line inputs. If the device has requested the intermittent high power option using General lingo "<a href="#">Command 0x13: IdentifyDeviceLingoes</a>" (page 80), it must wait until after this command is received before consuming more than 5 mA of accessory supply current. By the time the device receives this command, accessory high power is enabled and ready to use during the recording process.</p>
0x01	<p>End audio recording mode.</p> <p>When it receives this command, the device can deactivate its microphone recording inputs or outputs and return to a quiescent state. If the device has requested the intermittent high power option, by using the General lingo command <code>IdentifyDeviceLingoes</code>, it may be in a high power consumption state. After receiving this command, the device must reduce its power consumption below 5 mA of accessory supply current within 100 ms.</p>
0x02	<p>Begin audio playback mode.</p> <p>When it receives this command, the device can activate its speaker playback inputs or outputs connected to the iPod line outputs, if present. If the device has requested the intermittent high power option, by using the General lingo command <code>IdentifyDeviceLingoes</code>, it must wait until after this command is received before consuming more than 5 mA of accessory supply current. By the time the device receives this command, accessory high power is enabled and ready to use during the playback process.</p>

Value	Meaning
0x03	End audio playback mode.  When it receives this command, the device can deactivate its speaker playback inputs or outputs and return to a quiescent state. If the device has requested the intermittent high power option using the General lingo command <code>IdentifyDeviceLingoes</code> , it may be in a high power consumption state. After receiving this command, the device must reduce its power consumption below 5 mA of accessory supply current within 100 ms.
0x04–0xFF	Reserved.

**Note:** Failure to wait for a mode change notification before increasing power consumption could result in an incompatibility with present and future iPods.

Failure to reduce power consumption within the stated time could result in an incompatibility with present and future iPods.

## Command 0x07: GetDevCaps

Direction: iPod to Device

The iPod sends this command to the microphone device to determine the features present on the device. In response, the device sends "[Command 0x08: RetDevCaps](#)" (page 153) with the payload indicating the capabilities it supports.

The timeout for this command is 200 ms (0.2 second). The device must respond within the allotted time; the iPod will not retry the command. If the device does not respond within the specified time, the command will fail and the device may be considered not present by the iPod.

**Table 3-10** GetDevCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x07	Command ID: GetDevCaps
5	0xF6	Checksum

## Command 0x08: RetDevCaps

Direction: Device to iPod

The device sends this command in response to the command "[Command 0x07: GetDevCaps](#)" (page 153) sent by the iPod. The microphone device returns the payload indicating which capabilities it supports.

**Table 3-11** RetDevCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x08	Command ID: RetDevCaps
5	0xNN	Device capabilities (bits 31: 24). See <a href="#">Table 3-12</a> (page 154).
6	0xNN	Device capabilities (bits 23: 16)
7	0xNN	Device capabilities (bits 15: 8)
8	0xNN	Device capabilities (bits 7: 0)
9	0xNN	Checksum

The capabilities bit ranges correspond to the microphone control commands. The iPod should not attempt to control device features, using "[Command 0x0B: SetDevCtrl](#)" (page 156), if the associated capabilities bits are not set. [Table 3-12](#) (page 154) lists the meaning of these bits.

**Table 3-12** Microphone capabilities bitmask

Bit	Meaning
00	Stereo line input. A value of 0 indicates the device is monophonic only.
01	Stereo or mono line input. This bit should be set only if the microphone supports stereo line input and can switch between stereo and mono modes.
02	Recording level is present and variable.
03	Recording level limit is present.
04	Accessory supports duplex audio; it can play audio output from the iPod while it sends audio input to the iPod.
31:05	Reserved.

## Command 0x09: GetDevCtrl

Direction: iPod to Device

The iPod sends this command to get the device control state for the specified control type. In response, the device sends "[Command 0x0A: RetDevCtrl](#)" (page 155) with its current control state. If this command is not supported by the device—that is, if the microphone does not have any configurable controls—it must return an ACK command with a bad parameter error status.

The timeout for this command is 200 ms (0.2 second). The device must respond within the allotted time; the iPod will not retry the command. If the device does not respond within the specified time, the command will fail and the device may be considered not present by the iPod.

**Table 3-13** GetDevCtrl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x09	Command ID: GetDevCtrl
5	0xNN	The control type for which to get the state. The possible values are: 0x00: Reserved. 0x01: Stereo/mono line input. 0x02: Recording level control. 0x03: Recording level limiter control. 0x04–0xFF: Reserved.
6	0xNN	Checksum

## Command 0x0A: RetDevCtrl

Direction: Device to iPod

The device sends this command in response to the command "[Command 0x09: GetDevCtrl](#)" (page 154) received from the iPod. The device returns the current control state for the specified control type. Control types are supported only if the associated capabilities bits are set in the command "[Command 0x08: RetDevCaps](#)" (page 153).

**Table 3-14** RetDevCtrl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload

Byte number	Value	Comment
3	0x01	Lingo ID: Microphone lingo
4	0x0A	Command ID: RetDevCtrl
5	0xNN	The control type. See Table 3-15 (page 156).
6	0xNN	The control data. See Table 3-15 (page 156).
7	0xNN	Checksum

Table 3-15 (page 156) lists the different control types and the data associated with them.

**Table 3-15** Control types and data

Control type value	Control type	Control data
0x00	Reserved	Stereo capability cannot be set.
0x01	Stereo/mono line input	Possible data values are: 0x00 = mono 0x01 = stereo 0x02–0xFF = reserved
0x02	Recording level control	Possible data values are in a range between: 0x00 = mute 0xFF = maximum gain
0x03	Recording level limiter	Possible data values are: 0x00 = off 0x01 = on 0x02–0xFF = reserved
0x04–0xFF	Reserved	

## Command 0x0B: SetDevCtrl

Direction: iPod to Device

The iPod sends this command to set the device control state for the specified control type. In response, the device sends the **ACK** command with the command status. If this command is not supported by the device—that is, if the microphone does not have any configurable controls—it must return an **ACK** command with a bad parameter error status.

The timeout for this command is 200 ms (0.2 second). The device must respond within the allotted time; the iPod will not retry the command. If the device does not respond within the specified time, the command will fail and the device may be considered not present by the iPod.



**Table 3-16** SetDevCtrl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x0B	Command ID: SetDevCtrl
5	0xNN	The control type. The control type and data values are the same as for the RetDevCtrl (0x0A) command. See <a href="#">Table 3-15</a> (page 156) for more information.
6	0xNN	The control data. See <a href="#">Table 3-15</a> (page 156) for more information.
7	0xNN	Checksum

## Lingo 0x02: Simple Remote Lingo

This lingo is intended for a remote device that retains no state information about the iPod. Simple commands are sent to the iPod, and no acknowledgment or state information is sent back to the device. This lingo is used by the iPod standard in-line remote control. For a sample command sequence that declares this lingo, see [Table 3-272](#) (page 335).

## History and Applicability

System software versions 2.0 through 2.2 on the 3G iPod and versions 1.0 and 1.1 of the iPod mini support only the first five button responses (0 through 4) in the ContextButtonStatus command. This is called the contextual button lingo and includes only command 0x00. An extended set of commands (0x01 through 0x04) is available in the dedicated media lingoes, as shown in [Table 3-17](#) (page 157).

**Table 3-17** Simple remote lingo command summary

Command	ID	Direction	Packet	Lingo version	Authentication
0x00	ContextButtonStatus	Device to iPod	buttonStatus:4	All	No
0x01	ACK	iPod to Device	cmdStatus:1, cmdID:1	1.01	Yes
0x02	ImageButtonStatus	Device to iPod	buttonStatus:4	1.01	Yes
0x03	VideoButtonStatus	Device to iPod	buttonStatus:4	1.01	Yes
0x04	AudioButtonStatus	Device to iPod	buttonStatus:4	1.01	Yes

Command	ID	Direction	Packet	Lingo version	Authentication
0x05–0x0C	Reserved	N/A	N/A	N/A	N/A
0x0D	RadioButtonStatus	Device to iPod	buttonStatus:1	1.02	Yes
0x0E	CameraButtonStatus	Device to iPod	buttonStatus:1	1.02	Yes
0x0F–0xFF	Reserved	N/A	N/A	N/A	N/A

Dedicated media lingoes are supported by version 1.01 of the Simple Remote lingo. By using `GetiPodOptionsForLingo` (or querying the version number, as shown in [Table 3-18](#) (page 158) if the iPod does not support `GetiPodOptionsForLingo`), an accessory can determine the level of command support.

**Table 3-18** Simple remote lingo support versions

Version	Support
No version	Command 0 supported, buttons 0–4
No version	Command 0 supported, buttons 0–25
1.00	Version number can be obtained
1.01	Command 0–4 supported, buttons 0–25, media controls
1.02	Bug fix for compatibility with some accessories: when a device identifies itself using the General lingo <code>Identify</code> command, the 200 ms time limit during which <code>ContextButtonStatus</code> must report that all buttons are up is removed.

[Table 3-19](#) (page 158) shows the history of command changes in the Simple Remote lingo:

**Table 3-19** Simple Remote lingo command history

Lingo version	Command changes	Features
No version	Add: 0x00	Context specific button status, buttons 0–4
No version	None	Context-specific button status, buttons 5–25 added (4G iPod with firmware versions 3.0.0 and 3.0.1)
1.00	None	Version number available through <code>RequestLingoProtocol-Version</code>
1.01	Add: 0x01–0x04	Image-, video-, and audio-specific media control button status
1.02	Add: 0x0D, 0x0E	All-buttons-up timeout applied to all commands, except not to <code>ContextButtonStatus</code> when that command is used with the General lingo (0x00) <code>Identify</code> command (0x01)

## Playback Engine Playlists

---

The iPod **playback engine** contains a list of queued and currently playing media tracks. Its contents can be queried via the Display Remote or Extended Interface lingo to obtain track information such as track name, artist, album, genre, etc. One way the playback engine can be loaded is for the user to traverse the iPod UI, selecting one or more media menu items (such as music or videos) and pressing the play/pause button. Another way is for an accessory to send Extended Interface lingo commands to select database categories and initiate the playback of specific selections.

The list of media tracks currently queued for playback in the iPod's playback engine is called the **Now Playing** list. The playlist consisting of all tracks for a given media type, such as audio or video, is called the **All Tracks** list.

## Using Contextual Buttons

---

A simple remote device sends a `ContextButtonStatus` command to provide updated status on which buttons are held down or released. The data of the packet is a number of bytes indicating which buttons are currently held down. The bytes are constructed by ORing the masks of the buttons together. To indicate all buttons are released, the device must send a full data payload consisting of 0x00. While any buttons are held down, the device must periodically send an updated `ContextButtonStatus` packet on a 30 ms to 100 ms interval.

It is not necessary to transmit any trailing bytes in which no bits are set. If this option is exercised, the length of the packet in the header must be adjusted accordingly; that is, the packet payload length must be decreased to exclude the trailing zero byte(s) that will not be transmitted.

When the user presses and holds down a button, a simple remote device must generate the button status packet immediately and repeat it every 30 to 100 ms for as long as the button is pressed. If a second button is pressed while the first button is down, the button status packet sent by the device must include status for both buttons, and this packet must be repeated every 30 to 100 ms for as long as both buttons are held down. [Table 3-26](#) (page 168) lists the possible iPod button states.

The Next Track and Previous Track commands skip to the next or previous track. If the current track has played less than two seconds, Previous Track backs up to the beginning of the previous track. If the current track has played more than two seconds, it backs up to the beginning of the current track. These commands skip to the next or previous track even when the current track has chapters.

If the current track is an audiobook or a podcast with chapters, the Next Chapter and Previous Chapter commands skip to the next or previous chapter; otherwise they have no effect. If a track has chapters and has played more than two seconds of the current chapter, the Previous Chapter command backs up to the beginning of the current chapter; if the track has played less than two seconds, it backs up to the beginning of the previous chapter.

Some iPod button states are interpreted differently by the iPod when pressed and held down for 2 seconds or more. These are as follows:

- The Next Track button is treated as a Scan Forward button when pressed and held while a track is playing.
- The Previous Track button is treated as a Scan Backward button when pressed and held while a track is playing.
- The Play/Pause button is treated as a Power Off button when pressed and held.

- In iPods before the 4G iPod (color display), the Menu button acted as a Display Backlight On/Off button when pressed and held. Starting with the 4G iPod (color display), pressing and holding the Menu button causes a jump to the top level menu.
- If the iPod is in Browse mode, the Select button is treated as an Add Track to On-The-Go Playlist button when pressed and held.

Repeated Next Track and Previous Track commands (see [Table 3-26](#) (page 168)), without an intervening button status packet indicating all buttons are up, are interpreted as Fast Forward and Rewind commands. For a locking Fast Forward or Rewind button, use the Begin Fast Forward or Begin Rewind commands to start the operation and a Play/Resume command to return to the play state.

The Next and Previous Album commands (see [Table 3-26](#) (page 168)) have no effect if there is no next or previous album to go to in the Now Playing list. Similarly, the Next and Previous Chapter commands have no effect if the currently playing track does not have two or more chapters.

If the list of media tracks currently in the playback engine contains two or more different playlists (including the All Tracks playlist), the Next and Previous Playlist commands navigate these playlists. If the list of media tracks is from a single playlist, or none of the tracks are associated with a playlist, then the Next and Previous Playlist commands have no effect.

Use the following steps to wake an iPod when a simple remote button is pressed (UART serial port only):

1. Send a 0xFF sync byte.
2. Wait 20 ms.
3. Send the button status packet.
4. Wait 30 to 100 ms.
5. Repeat steps 3 and 4 for as long as any button is pressed.

Multiple button status packets cannot be sent back to back; otherwise, the repeated button status packets may be misinterpreted as being part of a corrupted packet. Repeated packets must always be separated by a gap of more than 25 ms, so the iPod knows that the new packet is not part of the previous packet (which may happen if the SYNC and SOP bytes in the first packet have been lost).

## Using Dedicated Media Buttons

---

The iAP contextual button protocol sends command packets with button messages that are interpreted based on the iPod user interface context. When an iPod is playing media types such as images and video, however, remote controls can become overloaded and their behavior may become confusing to the iPod user. In this case, the accessory device should use dedicated media control button commands.

The dedicated media lingoes include an ACK command, so devices know that a packet has been received, and dedicated button status commands for each media type currently supported by the iPod: images, slideshows, videos, and audio. Use of the dedicated media lingoes requires accessory device authentication.

Media control button status bits are organized so that the most frequently used buttons are assigned low bit positions. This reduces the button status packet sizes for frequently used buttons. Button status is maintained separately for all ports and all commands. This means that buttons can be in different states for different media control types.

**Note:** For a given port and media control type (except `ContextButtonStatus` from a device using the `Identify` command), if a command has not been received within approximately 200 ms after the last button status command, the button status will be reset to all buttons up.

## Accessory Control of the iPod 5G nano Camera

This section describes commands that let accessories control the video camera in the iPod 5G nano.

### iPod Camera Modes

The camera in the iPod 5G nano is operated by an iPod firmware application that can be on or off. In its Preview mode, the image that the camera captures is continuously displayed on the iPod's screen. Its Recording mode captures the video being previewed. The user can make each of three modes transition to other modes by means of user controls on either the iPod or its attached accessory, as shown in [Table 3-20](#) (page 161).

**Table 3-20** iPod camera modes and transitions

Mode	Description	Next mode	Transition control
Camera App Off	The iPod performs only non-camera functions.	Preview	iPod only
Preview	The iPod screen displays the image that can be captured as video. In this mode and Recording mode, the iPod's music capabilities are disabled.	Recording	iPod or accessory
		Camera App Off	iPod only
Recording	The iPod is recording video	Preview	iPod or accessory
		Camera App Off	iPod only

If a camera accessory is designed to receive feedback from the iPod, it must register for notifications, as specified in [Camera Accessory Identification](#) (page 162). After registration, the iPod notifies the accessory of its current mode and then sends notifications whenever it changes mode.

### Camera Sessions

An iPod camera session begins when the user launches the iPod's Camera Application. The application starts in Preview mode.

When the iPod is in Preview mode the user can order it to start recording video, using either the iPod or the accessory. The iPod goes to Recording mode and records video. When the user stops recording, using either the iPod or the accessory, the iPod returns to Preview mode.

In either Preview or Recording mode the user can quit the Camera Application, using the iPod. The iPod quits the application and returns to its non-camera functionality.

The following is a typical sequence of user actions to capture a video:

1. Launch the Camera application on the iPod. This sets Preview Mode.
2. Order video recording to start, using either the iPod or the accessory.

3. Order video recording to end, using either the iPod or the accessory, or quit the Camera Application, using the iPod.

## Accessory Feedback

In some situations, the user may operate the iPod's camera functions by means of an accessory without being able to see the iPod or otherwise get feedback from it. Thus it is desirable, but not required, for camera-control accessories to provide feedback to the user about the iPod camera's modes.

A simple feedback model might be for the accessory to provide a tristate indicator, such as a LED display, with OFF, MIDDLE, and ON states. When the accessory is connected to the iPod, and while it is going through its identification and authentication process, the indicator would be OFF. Its MIDDLE and ON states would then follow the guidelines shown in [Table 3-21](#) (page 162).

**Table 3-21** Suggested accessory feedback indications

iPod mode	Indicator	Comments
Camera App Off	OFF	Default accessory feedback state
Preview	MIDDLE	MIDDLE state indicates Preview mode
Recording	ON	Indicator remains ON until the iPod notifies the accessory that it has gone to Preview or Camera App Off mode, or communication between the accessory and the iPod is interrupted.

## Camera Accessory Identification

Every accessory that supports the iPod camera technology described in this document must identify and authenticate itself for at least the General and Simple Remote lingoes (Lingoes 0x00 and 0x02), as specified in ["Accessory Identification"](#) (page 445).

After it has received an `AckDevAuthenticationStatus` command with a status of 0x00 from the iPod (completing the authentication process), the accessory can start sending `CameraButtonStatus` commands to it. An accessory that wants feedback from the iPod may also query the iPod's notification capabilities, using `GetSupportedEventNotification` and `RetSupportedEventNotification`, then send a `SetEventNotification` command, as described in [Camera Interface Commands](#) (page 162). To get camera feedback, the `SetEventNotification` command sets bit 4 (Camera Notifications) in its bitmask. The iPod replies with an `iPodNotification` command, passing the current mode of its Camera Application. Because the user can set the iPod's camera to either Preview or Recording mode, the accessory must be prepared to configure itself to be compatible with any of the modes listed in [Table 3-20](#) (page 161).

If it has sent a `SetEventNotification` command, the accessory must be prepared to receive `iPodNotification` commands from the iPod every time the iPod's camera state changes. The accessory may use these notifications to provide feedback to the user, as suggested in [Accessory Feedback](#) (page 162).

## Camera Interface Commands

Accessory communication with the 5G nano camera is implemented by five iAP commands, as shown in [Table 3-22](#) (page 163).

**Table 3-22** iPod camera interface commands

Lingo ID	Cmd ID	Cmd name	Direction	Parameters
0x00	0x4F	GetSupportedEventNotification	Device to iPod	{transID:2}
0x00	0x51	RetSupportedEventNotification	iPod to Device	{transID:2,eventNotifyMask:8}
0x00	0x49	SetEventNotification	Device to iPod	{transID:2,eventNotifyMask:8}
0x00	0x4A	iPodNotification	iPod to Device	{transID:2,notificationType:1,payload:1}
0x02	0x0E	CameraButtonStatus	Device to iPod	{transID:2,buttonStatus:1}

## Sample Command Sequences

This section describes two scenarios in which an accessory communicates with a 5G nano to control its camera. For further information about notification commands, see [iPod Event Notifications](#) (page 449).

### Scenario 1: Video Control On and Off by the Accessory

A typical scenario for recording video under complete accessory control includes these actions:

1. The accessory is connected and authenticates itself. Its feedback indicator is OFF (see [Table 3-21](#) (page 162)).
2. The accessory sends `GetSupportedEventNotification` to the iPod to determine which event notifications it supports.
3. The iPod responds with `RetSupportedEventNotification`, passing a mask that reports which notifications it supports.
4. The accessory sends `SetEventNotification` to the iPod, requesting notifications for camera and flow control events.
5. The Camera Application is not yet launched, so the iPod sends the accessory an `iPodNotification` command with a payload of Camera App Off (see [Table 2-108](#) (page 132)).
6. Using iPod controls, the user launches the Camera application, which comes up in Preview mode.
7. The iPod sends the accessory an `iPodNotification` command with a payload of Preview.
8. The accessory changes its Indicator to the Middle state.
9. The user presses the camera button on the accessory.
10. The accessory sends a `CameraButtonStatus` command to the iPod, passing 0x01.
11. The accessory sends a second “button-up” `CameraButtonStatus` command to the iPod, passing 0x00.
12. The iPod transitions to its Recording mode and sends a corresponding notification to the accessory.

13. The accessory turns On its Indicator.
14. The iPod records video...
15. The user presses the camera button on the accessory.
16. The accessory sends a `CameraButtonStatus` command to the iPod, passing 0x01.
17. The accessory sends a second “button-up” `CameraButtonStatus` command to the iPod, passing 0x00. The iPod stops recording video.
18. The iPod transitions to its Preview mode and sends a corresponding notification to the accessory.
19. The accessory changes its Indicator to the Middle state.

As a result of this sequence, the accessory has started and ended video recording, during which time the accessory's Indicator has been On. A typical sequence of commands that implements this scenario is listed in [Table 3-23](#) (page 164).

**Table 3-23** Sample video control commands, accessory on and off

Step	Accessory command	iPod command	Comment
The accessory identifies itself as supporting the General and Simple Remote lingoes and performs the actions necessary to pass device authentication, as described in " <a href="#">Accessory Identification</a> " (page 445).			
1		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
2	GetSupportedEvent-Notification		requesting supported notifications
3		RetSupportedEvent-Notification	reporting that notifications for events 'flow control' and 'camera' are supported
4	SetEventNotification		setting notifications for events 'flow control' and 'camera'
5		iPodNotification	sending notification 'Camera Not Ready'
6		ACK	acknowledging 'Success (OK)' to command 'General Lingo::SetEventNotification'
The user launches the Camera application, using controls on the iPod.			
7		iPodNotification	sending notification 'Preview'
The user presses the Camera Action button on the accessory.			
8	CameraButtonStatus		Camera Action
9	CameraButtonStatus		button up



Step	Accessory command	iPod command	Comment
10		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
11		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
12		iPodNotification	sending notification 'Recording'
The user presses the Camera Action button on the accessory.			
13	CameraButtonStatus		Camera Action
14	CameraButtonStatus		button up
15		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
16		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
17		iPodNotification	sending notification 'Preview'

### Scenario 2: Video Control On by the iPod and Off by the Accessory

This scenario differs from Scenario 1 in that the iPod starts video recording and the accessory ends it. The following actions take place:

1. The accessory is connected and authenticates itself. Its feedback indicator is OFF (see [Table 3-21](#) (page 162)).
2. The accessory sends `GetSupportedEventNotification` to the iPod to determine which event notifications it supports.
3. The iPod responds with `RetSupportedEventNotification`, passing a mask that reports which notifications it supports.
4. The accessory sends `SetEventNotification` to the iPod, requesting notifications for camera and flow control events.
5. The Camera Application is not yet launched, so the iPod sends the accessory an `iPodNotification` command with a payload of Camera App Off (see [Table 2-108](#) (page 132)).
6. Using iPod controls, the user launches the Camera application, which starts in Preview mode.
7. The iPod sends the accessory an `iPodNotification` command with a payload of Preview.
8. The accessory changes its Indicator to the Middle state.
9. The user starts Recording using controls on the iPod.

10. The iPod sends the accessory an `iPodNotification` command with a payload of Recording.
11. The accessory turns On its Indicator.
12. The iPod records video...
13. The user presses the camera button on the accessory.
14. The accessory sends a `CameraButtonStatus` command to the iPod, passing 0x01.
15. The accessory sends a second “button-up” `CameraButtonStatus` command to the iPod, passing 0x00. The iPod stops recording video.
16. The iPod transitions to its Preview mode and sends a corresponding notification to the accessory.
17. The accessory changes its Indicator to the Middle state.

As a result of this sequence, the iPod has started and the accessory has ended a video recording, during which time the accessory's Indicator has been On. A typical sequence of commands that implements this scenario is listed in [Table 3-24](#) (page 166).

**Table 3-24** Sample video control commands, iPod on and accessory off

Step	Accessory command	iPod command	Comment
The accessory identifies itself as supporting the General and Simple Remote lingo and performs the actions necessary to pass device authentication, as described in <a href="#">"Accessory Identification"</a> (page 445).			
1		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
2	GetSupportedEvent-Notification		requesting supported notifications
3		RetSupportedEvent-Notification	reporting that notifications for events 'flow control' and 'camera' are supported
4	SetEventNotification		setting notifications for events 'flow control' and 'camera'
5		iPodNotification	sending notification 'Camera Not Ready'
6		ACK	acknowledging 'Success (OK)' to command 'General Lingo::SetEventNotification'
The user launches the Camera application, using controls on the iPod.			
7		iPodNotification	sending notification 'Preview'
The user starts recording video, using controls on the iPod.			
8		iPodNotification	sending notification 'Recording'

Step	Accessory command	iPod command	Comment
The user presses the Camera Action button on the accessory.			
9	CameraButtonStatus		Camera Action
10	CameraButtonStatus		button up
11		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
12		ACK	acknowledging 'Success (OK)' to command 'Simple Remote Lingo::CameraButtonStatus'
13		iPodNotification	sending notification 'Preview'

## Command 0x00: ContextButtonStatus

Direction: Device to iPod

The device sends this command to the iPod when a button event occurs. The button status is a bitmask representing each button that is currently pressed. The device must send the button status packet repeatedly at intervals between 30 and 100 ms, while one or more buttons are pressed. When all buttons are released, the device must send a button status packet with a 0x00 payload to indicate that no buttons are pressed. The iPod does not return a packet to the device in response to this command.

**Table 3-25** ContextButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03–0x06	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x00	Command ID: ContextButtonStatus
5	0xNN	Byte index 0, button states 7:0. See <a href="#">Table 3-26</a> (page 168) for a list of button states recognized by the iPod.
6	0xNN	Byte index 1, button states 15:8 (optional)
7	0xNN	Byte index 2, button states 23:16 (optional)
8	0xNN	Byte index 3, button states 31:24 (optional)
(last byte)	0xNN	Checksum

Table 3-26 (page 168) lists the available buttons and their bitmasks.

**Table 3-26** Button states

Button name	Number	Byte index	Button bitmask
Play/Pause	0	0x00	0x01
Volume Up	1	0x00	0x02
Volume Down	2	0x00	0x04
Next Track	3	0x00	0x08
Previous Track	4	0x00	0x10
Next Album	5	0x00	0x20
Previous Album	6	0x00	0x40
Stop	7	0x00	0x80
Play/Resume	8	0x01	0x01
Pause	9	0x01	0x02
Mute toggle	10	0x01	0x04
Next Chapter	11	0x01	0x08
Previous Chapter	12	0x01	0x10
Next Playlist	13	0x01	0x20
Previous Playlist	14	0x01	0x40
Shuffle Setting Advance	15	0x01	0x80
Repeat Setting Advance	16	0x02	0x01
Power On	17	0x02	0x02
Power Off	18	0x02	0x04
Backlight for 30 Seconds	19	0x02	0x08
Begin Fast Forward	20	0x02	0x10
Begin Rewind	21	0x02	0x20
Menu	22	0x02	0x40
Select	23	0x02	0x80
Up Arrow	24	0x03	0x01
Down Arrow	25	0x03	0x02

Button name	Number	Byte index	Button bitmask
Backlight Off	26	0x03	0x04
Reserved	27-31	0x03	0xF8

**Notes:**

- The Begin Fast Forward and Begin Rewind states must be sent every 30 to 100 ms to assure continuous fast forward or rewind actions.
- If the user holds down the Fast Forward button (sending repeated Fast Forward states) through a track change, the new track begins playing at normal speed.
- The Menu button navigates only within iPod/Music/Video applications and does not return the user to the Home screen.

## Command 0x01: ACK

---

Direction: iPod to Device

The iPod sends this command in response to any command sent from the device, except command 0x00. An ACK response is sent when a command that does not return any data has completed, when a bad parameter is received, or when an unsupported or invalid command is received.

**Table 3-27** ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x01	Command ID: ACK
5	0xNN	Status of command received (see <a href="#">Table 3-28</a> (page 169))
6	0xNN	ID of the command being acknowledged
7	0xNN	Checksum

**Table 3-28** Command status codes

Code	Command status
0x00	Command OK

Code	Command status
0x01	Unknown track category (not applicable)
0x02	Command failed (valid command, did not succeed)
0x03	Out of resources (iPod internal allocation failed)
0x04	Bad parameter (command or input parameters invalid)
0x05	Unknown track ID (not applicable)
0x06	Command pending ( <code>cmdPendTime</code> parameter returned)
0x07	Not authenticated (not authenticated)
0x08	Mismatched authentication protocol version
0x09–0xFF	Reserved

## Command 0x02: ImageButtonStatus

Direction: Device to iPod

The device sends this command to the iPod when an image-specific button event occurs. The button status is a bitmask representing each button that is currently pressed. The button status packet must be repeatedly sent by the device at intervals between 30 and 100 ms while one or more buttons are pressed. When all buttons are released, the device must send a button status packet with a 0x00 payload to indicate that no buttons are pressed. In response, the iPod will return an ACK packet containing the command status to the device.

**Table 3-29** ImageButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x02	Command ID: ImageButtonStatus
5	0xNN	Byte index 0, image-specific button states 7:0. See <a href="#">Table 3-30</a> (page 171) for a list of image-specific button states recognized by the iPod.
6	0xNN	Byte index 1, button states 15:8 (optional)
7	0xNN	Byte index 2, button states 23:16 (optional)
8	0xNN	Byte index 3, button states 31:24 (optional)

Byte number	Value	Comment
(last byte)	0xNN	Checksum

**Table 3-30** Image-specific button values

Button name	Number	Byte index	Button bitmask
Play/Pause	0	0x00	0x01
Next image	1	0x00	0x02
Previous image	2	0x00	0x04
Stop	3	0x00	0x08
Play/resume	4	0x00	0x10
Pause	5	0x00	0x20
Shuffle advance	6	0x00	0x40
Repeat advance	7	0x00	0x80
Reserved	8-15	0x01	0xFF
Reserved	16-23	0x02	0xFF
Reserved	24-31	0x03	0xFF

## Command 0x03: VideoButtonStatus

Direction: Device to iPod

The device sends this command to the iPod when a video-specific button event occurs. The button status is a bitmask representing each button that is currently pressed. The button status packet must be repeatedly sent by the device at intervals between 30 ms and 100 ms while one or more buttons are pressed. When all buttons are released, the device must send a button status packet with a 0x00 payload to indicate that no buttons are pressed. In response, the iPod will return an ACK packet containing the command status to the device.

**Table 3-31** VideoButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo

Byte number	Value	Comment
4	0x03	Command ID: <code>VideoButtonStatus</code>
5	0xNN	Byte index 0, video-specific button states 7:0. See <a href="#">Table 3-32</a> (page 172) for a list of video-specific button states recognized by the iPod.
6	0xNN	Byte index 1, button states 15:8 (optional)
7	0xNN	Byte index 2, button states 23:16 (optional)
8	0xNN	Byte index 3, button states 31:24 (optional)
(last byte)	0xNN	Checksum

**Table 3-32** Video-specific button values

Button name	Number	Byte index	Button bitmask
Play/Pause	0	0x00	0x01
Next video	1	0x00	0x02
Previous video	2	0x00	0x04
Stop	3	0x00	0x08
Play/Resume	4	0x00	0x10
Pause	5	0x00	0x20
Begin FF	6	0x00	0x40
Begin REW	7	0x00	0x80
Next chapter	8	0x01	0x01
Previous chapter	9	0x01	0x02
Next frame	10	0x01	0x04
Previous frame	11	0x01	0x08
Reserved	12-15	0x01	0xF0
Reserved	16-23	0x02	0xFF
Reserved	24-31	0x03	0xFF

The Next Video and Previous Video button commands skip to the next or previous video. If the current video has played less than two seconds, Previous Video backs up to the beginning of the previous video; if it has played more than two seconds, it backs up to the beginning of the current video. These commands skip to the next or previous video even when the current video has chapters. If the current video has chapters, the Next Chapter and Previous Chapter commands skip to the next or previous chapter; otherwise they have no



effect. If the video has chapters and has played more than two seconds of the current chapter, the Previous Chapter command backs up to the beginning of the current chapter; if the track has played less than two seconds, it backs up to the beginning of the previous chapter.

## Command 0x04: AudioButtonStatus

Direction: Device to iPod

The device sends this command to the iPod when an audio-specific button event occurs. The button status is a bitmask representing each button that is currently pressed. The button status packet must be repeatedly sent by the device at intervals between 30 ms and 100 ms while one or more buttons are pressed. When all buttons are released, the device must send a button status packet with a 0x00 payload to indicate that no buttons are pressed. In response, the iPod will return to the device an ACK message containing the command status.

**Table 3-33** AudioButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x04	Command ID: AudioButtonStatus
5	0xNN	Byte index 0, audio-specific button states 7:0. See <a href="#">Table 3-34</a> (page 173) for a list of audio-specific button states recognized by the iPod.
6	0xNN	Byte index 1, button states 15:8 (optional)
7	0xNN	Byte index 2, button states 23:16 (optional)
8	0xNN	Byte index 3, button states 31:24 (optional)
(last byte)	0xNN	Checksum

**Table 3-34** Audio-specific button values

Button name	Number	Byte index	Button bitmask
Play/Pause	0	0x00	0x01
Volume Up	1	0x00	0x02
Volume Down	2	0x00	0x04
Next Track	3	0x00	0x08
Previous Track	4	0x00	0x10

Button name	Number	Byte index	Button bitmask
Next Album	5	0x00	0x20
Previous Album	6	0x00	0x40
Stop	7	0x00	0x80
Play/Resume	8	0x01	0x01
Pause	9	0x01	0x02
Mute toggle	10	0x01	0x04
Next chapter	11	0x01	0x08
Previous chapter	12	0x01	0x10
Next playlist	13	0x01	0x20
Previous playlist	14	0x01	0x40
Shuffle setting advance	15	0x01	0x80
Repeat setting advance	16	0x02	0x01
Begin FF	17	0x02	0x02
Begin REW	18	0x02	0x04
Record	19	0x02	0x08
Reserved	20-23	0x02	0xF0
Reserved	24-31	0x03	0xFF

The Next and Previous Track, Album, and Playlist button commands skip to the next or previous track, album, or playlist. If the current track, album, or playlist has played less than two seconds, the Previous command backs up to the beginning of the previous one; if it has played more than two seconds, it backs up to the beginning of the current one. These commands skip to the next or previous track, album, or playlist even when the current one has chapters. If the current track has chapters, the Next Chapter and Previous Chapter commands skip to the next or previous chapter; otherwise they have no effect. If the track has chapters and has played more than two seconds of the current chapter, the Previous Chapter command backs up to the beginning of the current chapter; if the track has played less than two seconds, it backs up to the beginning of the previous chapter.

## Command 0x0D: RadioButtonStatus

Direction: Device to iPod

The accessory sends this command to a 5G nano iPod to initiate a tagging action by the iPod's Radio Application. Once the accessory has authenticated itself for the Simple Remote lingo, it may send this command at any time.

Currently the only payloads for this command are the values 0x01 and 0x00 in `buttonStatus` (byte 7). Any other values will result in the iPod returning an error value of 0x04 (Bad Parameter). The accessory should repeatedly send this command with a payload of 0x01 at intervals of 30-100 ms while its Radio Tag button is held down. When the button is released it must send a payload of 0x00 to indicate that no buttons are down. The iPod responds with an `ACK` command passing success (0x00).

This command is intended to be used with the `GetSupportedEventNotification` and `SetEventNotification` commands; see [iPod Event Notifications](#) (page 449).

**Table 3-35** `RadioButtonStatus` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x0D	Command ID: <code>RadioButtonStatus</code>
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of this command.
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>buttonStatus</code> : 0x01 = Radio button pushed to tag current song; 0x00 = button released.
8	0xNN	Checksum

## Command 0x0E: `CameraButtonStatus`

Direction: Device to iPod

The accessory sends this command to the iPod to initiate changes in the iPod's Camera mode. Once the accessory has authenticated itself for the Simple Remote lingo, it may send this command at any time.

The accessory must send `CameraButtonStatus` commands in pairs: the first command must pass 0x01 in `buttonStatus` (byte 7) and the second command must pass 0x00. If the accessory doesn't send a second command, the iPod assumes such a return after a 200 ms timeout. Each `CameraButtonStatus` command may pass only the value 0x00 or 0x01; otherwise, the iPod will acknowledge it with a Bad Parameter error status. The iPod will acknowledge each valid `CameraButtonStatus` command with a General lingo `ACK` command reporting Status OK (0x00).

This command is intended to be used with the `GetSupportedEventNotification` and `SetEventNotification` commands; see [iPod Event Notifications](#) (page 449).

**Table 3-36** CameraButtonStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x02	Lingo ID: Simple Remote lingo
4	0x0E	Command ID: CameraButtonStatus
5	0xNN	transID [bits 15:8]: Transaction ID of this command.
6	0xNN	transID [bits 7:0]
7	0xNN	buttonStatus: (0x01 = down, 0x00 = up)
8	0xNN	Checksum

## Lingo 0x03: Display Remote Lingo

The Display Remote lingo is for accessory devices that need to control the state of the iPod, display information about the state of the iPod on a remote display, or control the state of the iPod equalizer. The Display Remote protocol can be used by simple inline-display remotes (remotes that have single-line display and play control buttons) and more complex devices that have full multiline graphical displays to show information about the track, artist, or album; current play or pause state; track position; battery; shuffle and time.

For example, the Display Remote protocol can be used in an automotive application to show currently playing track information on the in-vehicle display while allowing the user to browse the database stored in the iPod. Such an application could let the user choose where to browse the database (on the in-vehicle display or on the iPod) and switch between Extended Interface and Display Remote modes, depending on the setting. For a sample command sequence that declares this lingo, see [Table 3-272](#) (page 335).

By supporting multiple lingoes, an accessory can use the Display Remote lingo in combination with other lingoes to create a fully functional iPod/accessory system. Accessories can also use this lingo to control the state of the iPod equalizer. The Display Remote lingo supports serial accessories attached to the 30-pin connector.

The Display Remote command set uses a single byte command format similar to the General and Simple Remote lingoes. Devices using the Display Remote lingo can identify using the General lingo (0x0) command `IdentifyDeviceLingoes` (0x13). See "[Command 0x13: IdentifyDeviceLingoes](#)" (page 80) for more information.

**Note:** The Display Remote lingo is an authenticated lingo, with some exceptions. USB accessories must authenticate before they can use the Display Remote lingo. Serial accessories have access only to the equalizer control, battery state, and sound check state commands (commands 0x01–0x07 and 0x1A–0x1E) if they do not authenticate. Please refer to [Table 3-37](#) (page 177) to determine which commands require authentication.

The Display Remote lingo can operate in notification (interrupt) mode, where the iPod sends event notifications to the device, or in polled (non-interrupt) mode. In polled mode, the device should send requests for state change information to the iPod.

The Display Remote commands export text as UTF-8 characters. Graphics cannot be exported. Note that Chapter count information can be retrieved only from the currently playing track.

**Table 3-37** Display Remote lingo command summary

Command	ID	Data length	Protocol version	Requires authentication over serial link
ACK	0x00	0x02	1.00	No
GetCurrentEQProfileIndex	0x01	0x00	1.00	No
RetCurrentEQProfileIndex	0x02	0x04	1.00	No
SetCurrentEQProfileIndex	0x03	0x05	1.00	No
GetNumEQProfiles	0x04	0x00	1.00	No
RetNumEQProfiles	0x05	0x04	1.00	No
GetIndexedEQProfileName	0x06	0x04	1.00	No
RetIndexedEQProfileName	0x07	0xNN	1.00	No
SetRemoteEventNotification	0x08	0x04	1.02	Yes
RemoteEventNotification	0x09	0xNN	1.02	Yes
GetRemoteEventStatus	0x0A	0x00	1.02	Yes
RetRemoteEventStatus	0x0B	0x04	1.02	Yes
GetiPodStateInfo	0x0C	0x01	1.02	Yes
RetiPodStateInfo	0x0D	0xNN	1.02	Yes
SetiPodStateInfo	0x0E	0xNN	1.02	Yes
GetPlayStatus	0x0F	0x00	1.02	Yes
RetPlayStatus	0x10	0x0D	1.02	Yes

Command	ID	Data length	Protocol version	Requires authentication over serial link
SetCurrentPlayingTrack	0x11	0x04	1.02	Yes
GetIndexedPlayingTrackInfo	0x12	0x07	1.02	Yes
RetIndexedPlayingTrackInfo	0x13	0xNN	1.02	Yes
GetNumPlayingTracks	0x14	0x00	1.02	Yes
RetNumPlayingTracks	0x15	0x04	1.02	Yes
GetArtworkFormats	0x16	0x02	1.04	Yes
RetArtworkFormats	0x17	0xNN	1.04	Yes
GetTrackArtworkData	0x18	0x0C	1.04	Yes
RetTrackArtworkData	0x19	0xNN	1.04	Yes
GetPowerBatteryState	0x1A	0x00	1.02	No
RetPowerBatteryState	0x1B	0x02	1.02	No
GetSoundCheckState	0x1C	0x00	1.02	No
RetSoundCheckState	0x1D	0x01	1.02	No
SetSoundCheckState	0x1E	0x02	1.02	No
GetTrackArtworkTimes	0x1F	0x0C	1.04	Yes
RetTrackArtworkTimes	0x20	0xNN	1.04	Yes
Reserved	0x21–0xFF	N/A	N/A	N/A

## Command History of the Display Remote lingo

Table 3-38 (page 178) shows the history of command changes in the Display Remote lingo:

**Table 3-38** Display Remote lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x07	iPod Equalizer Setting save/restore control
1.01	None	BugFix: Equalizer state not restored on extended interface exit
1.02	Add: 0x08–0x15, 0x1A–0x1E	Event notifications, iPod state info, playback track info, sound check, power/battery support

Lingo version	Command changes	Features
1.03	None	BugFix: Fix intermittent UI hang when restoring on exit
1.04	Add: 0x16–0x19, 0x1F–0x20	Track artwork, lyrics, track time position in seconds
1.05	None	Video browsing support added to playback commands. Chapter information can be retrieved for all tracks in the Now Playing list, not just for the currently playing track.

## Transferring Album Art

The Display Remote lingo includes several commands that support the transfer of album artwork from an iPod to an accessory:

- `GetArtworkFormats`
- `RetArtworkFormats`
- `GetTrackArtworkTimes`
- `RetTrackArtworkTimes`
- `GetTrackArtworkData`
- `RetTrackArtworkData`

All album art image encoding is RGB-565, which can be transferred in both big- and small-endian formats. Artwork retrieval takes place in the following steps:

1. Retrieve the number of formats available for artwork on the iPod using `GetArtworkFormats`. It is not necessary to call `GetArtworkFormats` more than once per session; these values will be static while the accessory is attached to the iPod. However, there are no guarantees about the number of formats and which ones are available on a particular model or firmware version. Each `formatID` in `RetArtworkFormats` specifies both a pixel encoding, such as RGB-565 little-endian, and the image dimensions. All formats are fixed-size.
2. When the accessory wants to retrieve the artwork for a given track, it calls `GetIndexedPlayingTrackInfo` with an `infoType` of 0x08. This returns the count of artwork available for each `formatID` associated with the track. It is possible that a track may not have artwork for a particular `formatID` or that the number of images will vary by `formatID`. A given size of album artwork may be available for only one track in the database.
3. To retrieve the list of images associated with a given track and `formatID`, the accessory calls `GetTrackArtworkTimes`. This command tells the iPod to return the associated timestamp for each artwork. The timestamp indicates when the artwork should be displayed, expressed in milliseconds from the start of playback.
4. When the accessory wants to retrieve an individual piece of artwork, it sends `GetTrackArtworkData` to the iPod. This requires the accessory to specify a track, a `formatID`, and the timestamp of the desired image. The iPod returns the specified artwork and the accessory can display it whenever it chooses.

## Command 0x00: ACK

Direction: iPod to Device

The iPod sends this command to acknowledge the receipt of a command from the device and return the command status. The command ID field indicates the device command for which the response is being sent. The command status indicates the result of the command (success or failure).

**Table 3-39** ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x00	Command ID: ACK
5	0xNN	Command result status. See <a href="#">Table 3-40</a> (page 180).
6	0xNN	The ID for the command being acknowledged.
7	0xNN	Checksum

[Table 3-40](#) (page 180) lists the possible result values for the command result status field.

**Table 3-40** Command result values

Result	Meaning
0x00	Success (OK)
0x01	Reserved
0x02	ERROR: Command failed
0x03	ERROR: Out of resources
0x04	ERROR: Bad parameter
0x05	ERROR: Unknown ID
0x06	Reserved
0x07	ERROR: Accessory not authenticated
0x08–0xFF	Reserved



## Command 0x01: GetCurrentEQProfileIndex

Direction: Device to iPod

The device requests the current Equalizer Profile setting index. In response, the iPod sends the "[Command 0x02: RetCurrentEQProfileIndex](#)" (page 181) packet.

**Table 3-41** GetCurrentEQProfileIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x01	Command ID: GetCurrentEQProfileIndex
5	0xFA	Checksum

## Command 0x02: RetCurrentEQProfileIndex

Direction: iPod to Device

The iPod sends this command, returning the current Equalizer Profile setting index, in response to the "[Command 0x01: GetCurrentEQProfileIndex](#)" (page 181) packet sent by the device. An Equalizer Index of 0x0 indicates that the equalizer is disabled.

**Table 3-42** RetCurrentEQProfileIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x02	Command ID: RetCurrentEQProfileIndex
5	0xNN	Current Equalizer Index (bits 31:24)
6	0xNN	Current Equalizer Index (bits 23:16)
7	0xNN	Current Equalizer Index (bits 15:8)
8	0xNN	Current Equalizer Index (bits 7:0)

Byte number	Value	Comment
9	0xNN	Checksum

## Command 0x03: SetCurrentEQProfileIndex

Direction: Device to iPod

The device sets the current Equalizer Profile setting index and optionally restores the original Equalizer Setting on accessory detach. The valid Equalizer Index range can be determined by sending "[Command 0x04: GetNumEQProfiles](#)" (page 183). An Equalizer Index of 0x0 tells the iPod that the equalizer should be disabled; a valid nonzero index enables the equalizer.

In response to this command, the iPod returns an ACK packet with the status of this command.

**Table 3-43** SetCurrentEQProfileIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x03	Command ID: SetCurrentEQProfileIndex
5	0xNN	Current Equalizer Index (bits 31:24)
6	0xNN	Current Equalizer Index (bits 23:16)
7	0xNN	Current Equalizer Index (bits 15:8)
8	0xNN	Current Equalizer Index (bits 7:0)
9	0xNN	bRestoreOnExit. Specifies whether to restore the previous Equalizer Setting on device exit or detach. See the discussion below.
10	0xNN	Checksum

The bRestoreOnExit Boolean byte flag determines the behavior of the iPod when the accessory device is detached from the connector. A value of 0x0 (false) indicates that the original Equalizer Setting should be discarded. A nonzero (true) value indicates that the previous Equalizer Setting should be restored when the device is detached from the iPod. Anytime the SetCurrentEQProfileIndex command is sent with bRestoreOnExit equal to false, the previous equalizer state is erased and lost. If SetCurrentEQProfileIndex is sent with bRestoreOnExit equal to true every time, the first SetCurrentEQProfileIndex command saves the original equalizer state; subsequent commands do not change the original saved equalizer state. On accessory detach, the original saved equalizer state is restored.

## Command 0x04: GetNumEQProfiles

Direction: Device to iPod

The device requests the number of iPod Equalizer Profile settings. In response, the iPod sends the "[Command 0x05: RetNumEQProfiles](#)" (page 183) packet.

**Table 3-44** GetNumEQProfiles packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x04	Command ID: GetNumEQProfiles
5	0xF7	Checksum

## Command 0x05: RetNumEQProfiles

Direction: iPod to Device

The iPod returns the number of Equalizer Profiles in it. It sends this command in response to the "[Command 0x04: GetNumEQProfiles](#)" (page 183) packet sent by the device. The valid profile index range for iPod equalizer commands accepting a profile index is 0x0 to `profileCount-1`.

**Table 3-45** RetNumEQProfiles packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x05	Command ID: RetNumEQProfiles
5	0xNN	<code>profileCount</code> . The Equalizer Profile count (bits 31:24).
6	0xNN	Equalizer profile count (bits 23:16)
7	0xNN	Equalizer profile count (bits 15:8)
8	0xNN	Equalizer profile count (bits 7:0)

Byte number	Value	Comment
9	0xNN	Checksum

## Command 0x06: GetIndexedEQProfileName

---

Direction: Device to iPod

The device requests the iPod Equalizer Profile setting name for a given Equalizer Profile index. In response, the iPod sends the "[Command 0x07: RetIndexedEQProfileName](#)" (page 184) packet. The valid profile index range can be obtained by sending "[Command 0x04: GetNumEQProfiles](#)" (page 183).

**Table 3-46** GetIndexedEQProfileName packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x06	Command ID: GetIndexedEQProfileName
5	0xNN	Equalizer profile index (bits 31:24)
6	0xNN	Equalizer profile index (bits 23:16)
7	0xNN	Equalizer profile index (bits 15:8)
8	0xNN	Equalizer profile index (bits 7:0)
9	0xNN	Checksum

## Command 0x07: RetIndexedEQProfileName

---

Direction: iPod to Device

The iPod returns its Equalizer Profile setting name for the specified Equalizer Profile index in response to "[Command 0x06: GetIndexedEQProfileName](#)" (page 184). The Equalizer Profile name is returned as a variable-length, null-terminated UTF-8 character array.

**Note:** The UTF-8 Equalizer Profile name string is not limited to 255 characters. It may be sent in either small or large packet format. The following table shows the small packet format.

**Table 3-47** RetIndexedEQProfileName packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x07	Command ID: RetIndexedEQProfileName
5 ... N	0xNN...	The Equalizer Profile name, as a null-terminated UTF-8 character array.
(last byte)	0xNN	Checksum

## Command 0x08: SetRemoteEventNotification

Direction: Device to iPod

The device requests that the iPod enable asynchronous remote event notification for specific iPod events. Notification for each event can be enabled by setting the associated bit in the remote event bitmask (`remEventMask`). By default, all event notifications are disabled and must be explicitly enabled using this command. In response, the iPod sends an ACK command indicating the command completion status. A remote event bitmask of 0x0 disables all remote event status notifications. On device detach, event notification is reset to the default disabled state.

**Table 3-48** SetRemoteEventNotification packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x08	Command ID: SetRemoteEventNotification
5	0xNN	<code>remEventMask</code> (bits 31:24). See <a href="#">Table 3-49</a> (page 186) for a list of the events for which you can enable notification.
6	0xNN	<code>remEventMask</code> (bits 23:16)
7	0xNN	<code>remEventMask</code> (bits 15:8)

Byte number	Value	Comment
8	0xNN	remEventMask (bits 7:0)
9	0xNN	Checksum

Enable notifications for the events listed in [Table 3-49](#) (page 186) by setting the bit for each event in the remote event bitmask. A value of 1 enables the notification of the iPod state change for that event and a value of 0 disables the notification.

**Table 3-49** iPod events

Bit number	Remote Event
0	Track time position in milliseconds
1	Track playback index
2	Chapter index
3	Play status (play, pause, stop, FF, and RW)
4	Mute/UI Volume
5	Power/battery
6	Equalizer setting
7	Shuffle setting
8	Repeat setting
9	Date and time setting
10	Alarm setting
11	Backlight state
12	Hold switch state
13	Sound check state
14	Audiobook speed
15	Track time position in seconds
16	Mute/UI/Absolute Volume (see Note below)
31:17	Reserved

**Note:** When bit 16 is set in the remote event bitmask of `SetRemoteEventNotification`, `RemoteEventNotification` returns the same mute state and UI volume level as for bit 4. The absolute volume value it returns is the actual audio volume after the iPod volume limit has been applied. Its range is 0 to 255 if the iPod volume limit setting is set to maximum scale; otherwise its range is 0 to iPod volume limit. The UI volume level is scaled to cover the absolute volume range. This lets an accessory with a headphone jack comply with the iPod volume limit setting.

## Command 0x09: RemoteEventNotification

Direction: iPod to Device

The iPod sends this command asynchronously whenever an enabled event change has occurred. Use "[Command 0x08: SetRemoteEventNotification](#)" (page 185) to control which events are enabled. The notification packet formats are described in more detail below. Notifications for enabled events are sent every 500 ms, with the exception of volume change notifications, which are sent every 100 ms.

**Note:** Notifications are sent only when the iPod is in Power On mode; none are sent when the iPod is in Sleep or Hibernate mode.

This is the command packet for the `RemoteEventNotification` command. See [Table 3-51](#) (page 188) to interpret the `eventNum` and `eventData` fields.

**Table 3-50** `RemoteEventNotification` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x09	Command ID: <code>RemoteEventNotification</code>
5	0xNN	<code>eventNum</code> . A number indicating the type of event.
6 ... N	0xNN	<code>eventData</code> . Additional information about the event. This field is variable length; see <a href="#">Table 3-51</a> (page 188).
(last byte)	0xNN	Checksum

[Table 3-51](#) (page 188) shows the expected payload and length for each type of event notification. For example, if an accessory receives an event notification for event 0x02 (Chapter Info) then the payload is 8 bytes long and contains the track index (bytes 0–3), the chapter count (bytes 4 and 5) and the chapter index (bytes 6 and 7). If bytes 4 and 5 are all zeros, the currently playing track does not have chapters.

**Table 3-51** Event notification data

Event number	Event	Event data	Data length in bytes
0x00	Track position	<p>The new track position, in milliseconds.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Track Position (bits 31:24)</li> <li>■ Byte 1: Track Position (bits 23:16)</li> <li>■ Byte 2: Track Position (bits 15:8)</li> <li>■ Byte 3: Track Position (bits 7:0)</li> </ul>	0x04
0x01	Track index	<p>The index of the currently playing track.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Track Index (bits 31:24)</li> <li>■ Byte 1: Track Index (bits 23:16)</li> <li>■ Byte 2: Track Index (bits 15:8)</li> <li>■ Byte 3: Track Index (bits 7:0)</li> </ul>	0x04
0x02	Chapter information	<p>Chapter information, including the track index, chapter count, and chapter index.</p> <ul style="list-style-type: none"> <li>■ Bytes 0–3 specify the currently playing track index, as follows: <ul style="list-style-type: none"> <li>Byte 0: Track Index (bits 31:24)</li> <li>Byte 1: Track Index (bits 23:16)</li> <li>Byte 2: Track Index (bits 15:8)</li> <li>Byte 3: Track Index (bits 7:0)</li> </ul> </li> <li>■ Bytes 4–5 specify the chapter count of the track, as follows. A value of 0x0000 indicates that the track does not have chapters. <ul style="list-style-type: none"> <li>Byte 4: Chapter Count (bits 15:8)</li> <li>Byte 5: Chapter Count (bits 7:0)</li> </ul> </li> <li>■ Bytes 6–7 specify the chapter index, as follows. A value of 0xFFFF indicates that the track does not have chapters. <ul style="list-style-type: none"> <li>Byte 6: Chapter Index (bits 15:8)</li> <li>Byte 7: Chapter Index (bits 7:0)</li> </ul> </li> </ul>	0x08
0x03	Play status	<p>The current play status of the iPod; that is, whether it is playing, paused, stopped, fast forwarding or rewinding. Possible values are described in <a href="#">Table 3-52</a> (page 191).</p> <ul style="list-style-type: none"> <li>■ Byte 0: Play Status (bits 7:0)</li> </ul>	0x01



Event number	Event	Event data	Data length in bytes
0x04	Mute/UI Volume	<p>The current state of the mute setting and UI volume information.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Mute State (bits 7:0) A value of 0 indicates that mute is off; a value of 1 indicates that mute is on.</li> <li>■ Byte 1: UI Volume Level (bits 7:0) A value between 0 and 255, with 0 indicating minimum volume and 255 indicating maximum volume.</li> </ul> <p>Note that if the Mute State value is true (mute is on), the UI volume level field is not valid and is returned as 0.</p>	0x02
0x05	Power/battery	<p>Information about the power and battery status.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Power State (bits 7:0) A value indicating the current power source and its state. See <a href="#">Table 3-55</a> (page 192) for possible values.</li> <li>■ Byte 1: Battery Level (bits 7:0) Specifies the current battery level. A value from 0 to 255, with 0 indicating a fully discharged battery and 255 indicating a battery that is fully charged.</li> </ul> <p>If an external power status is returned, the battery level is invalid and is returned as 0.</p>	0x02
0x06	Equalizer state	<p>The current Equalizer Setting index.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Equalizer index (bits 31:24)</li> <li>■ Byte 1: Equalizer index (bits 23:16)</li> <li>■ Byte 2: Equalizer index (bits 15:8)</li> <li>■ Byte 3: Equalizer index (bits 7:0)</li> </ul>	0x04
0x07	Shuffle	<p>The state of the shuffle setting. See <a href="#">Table 3-53</a> (page 192) for a list of possible values.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Shuffle State (bits 7:0)</li> </ul>	0x01
0x08	Repeat	<p>The state of the repeat setting. See <a href="#">Table 3-54</a> (page 192) for a list of possible values.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Repeat State (bits 7:0)</li> </ul>	0x01

Event number	Event	Event data	Data length in bytes
0x09	Date/time	<p>The current date and time.</p> <ul style="list-style-type: none"> <li>■ Bytes 0–1 specify the current year. A value of 2005 represents the year 2005 A.D. <ul style="list-style-type: none"> <li>Byte 0: Year (bits 15:8)</li> <li>Byte 1: Year (bits 7:0)</li> </ul> </li> <li>■ Byte 2: Month (bits 7:0) <ul style="list-style-type: none"> <li>A value between 1 and 12, where 1 = January and 12 = December.</li> </ul> </li> <li>■ Byte 3: Day of the month (bits 7:0) <ul style="list-style-type: none"> <li>A value between 1 and 31.</li> </ul> </li> <li>■ Byte 4: Hour (bits 7:0) <ul style="list-style-type: none"> <li>A value between 0 and 23, where 0 = 12:00 a.m. and 23 = 11:00 p.m.</li> </ul> </li> <li>■ Byte 5: Minute (bits 7:0) <ul style="list-style-type: none"> <li>A value between 0 and 59.</li> </ul> </li> </ul>	0x06
0x0A	Alarm	<p>Alarm information.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Alarm State (bits 7:0) <ul style="list-style-type: none"> <li>A value indicating the current state of the alarm. Possible values are: <ul style="list-style-type: none"> <li>0 = off</li> <li>1 = enabled</li> <li>2 = triggered</li> </ul> </li> </ul> </li> <li>■ Byte 1: Alarm Hour (bits 7:0) <ul style="list-style-type: none"> <li>A value between 0 and 23, where 0 = 12:00 a.m. and 23 = 11:00 p.m.</li> </ul> </li> <li>■ Byte 2: Alarm Minute (bits 7:0) <ul style="list-style-type: none"> <li>A value between 0 and 59.</li> </ul> </li> </ul>	0x03
0x0B	Backlight	<p>The current backlight level. A value between 0 and 255, where 0 indicates the backlight is off and 255 indicates that the backlight is at full intensity.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Backlight Level (bits 7:0)</li> </ul>	0x01
0x0C	Hold switch	<p>The current state of the hold switch. A value of 0 means the hold switch is off. A value of 1 indicates it is on.</p> <ul style="list-style-type: none"> <li>■ Byte 0: Hold Switch State (bits 7:0)</li> </ul>	0x01

Event number	Event	Event data	Data length in bytes
0x0D	Sound check	The state of the sound check setting. A value of 0 means that sound check is off; a value of 1 indicates it is on. ■ Byte 0: Sound Check State (bits 7:0)	0x01
0x0E	Audiobook	The audiobook playback speed setting. See <a href="#">Table 3-56</a> (page 193) for a list of possible values. ■ Byte 0: Audiobook Playback Speed Setting (bits 7:0)	0x01
0x0F	Track position in seconds	The new track time position, in seconds. ■ Byte 0: Track Position (bits 15:8) ■ Byte 1: Track Position (bits 7:0)	0x02
0x10	Mute/UI/Absolute Volume	The current state of the mute setting, UI volume, and absolute volume. ■ Byte 0: Mute state (bits 7:0). A value of 0 indicates that muting is off; a value of 1 indicates that muting is on. ■ Byte 1: UI volume level (bits 7:0). A value between 0 and 255, normalized to volume limit settings. 0 indicates minimum volume and 255 indicates maximum volume. ■ Byte 2: Absolute volume level (bits 7:0). A value between 0 and 255, not normalized. 0 indicates minimum absolute volume and 255 indicates maximum absolute volume. If muting is on (byte 0 = 0x00), the absolute volume level is not valid and is returned as 0.	0x02
0x11–0xFF	Reserved	N/A	N/A

The battery and volume UI levels (but not absolute levels) are normalized across all iPod platforms so that 0 represents the minimum level and 255 represents the maximum level. The granularity of minimum to maximum range steps varies by iPod platform.

[Table 3-52](#) (page 191) lists the possible values for the data associated with a Play Status event and their meanings.

**Table 3-52** Play status values

Value	Meaning
0x00	Playback stopped
0x01	Playing (for " <a href="#">Command 0x0E: SetiPodStateInfo</a> " (page 197), start or resume playback)
0x02	Playback paused
0x03	Fast forward (FF)

Value	Meaning
0x04	Fast rewind (REW)
0x05	End fast forward or rewind mode
0x06–0xFF	Reserved

**Note:** With the iPhone, incoming phone calls can pause audio playback at any time. The accessory should enable notifications for this event and act accordingly (for example, by changing an icon). It should not try to cancel the pause.

Table 3-53 (page 192) lists the possible values for the data associated with a Shuffle event.

**Table 3-53** Shuffle state

Value	Meaning
0x00	Shuffle off
0x01	Shuffle tracks and songs
0x02	Shuffle albums
0x03–0xFF	Reserved

Table 3-54 (page 192) lists the possible values for the data associated with a Repeat event.

**Table 3-54** Repeat state

Value	Meaning
0x00	Repeat off
0x01	Repeat one track or song
0x02	Repeat all tracks
0x03–0xFF	Reserved

Table 3-55 (page 192) lists the possible values for the data associated with a Power/Battery event and their meanings.

**Table 3-55** Power and battery state

Value	Meaning
0x00	Internal battery power, low power (< 30%)
0x01	Internal battery power

Value	Meaning
0x02	External power, battery pack, no charging
0x03	External power, no charging
0x04	External power, battery charging
0x05	External power, battery charged
0x06–0xFF	Reserved

[Table 3-56](#) (page 193) lists the possible values for the data associated with an Audiobook event and their meanings.

**Table 3-56** Audiobook playback speeds

Value	Meaning
0xFF	Slower (–1)
0x00	Normal
0x01	Faster (+1)
0x02–0xFE	Reserved

## Command 0x0A: GetRemoteEventStatus

Direction: Device to iPod

The device requests the status of state information that has changed on the iPod. In response, the iPod sends "[Command 0x0B: RetRemoteEventStatus](#)" (page 194), containing a bitmask of event states that changed since the last `GetRemoteEventStatus` command and clears all the remote event status bits. This command may be used to poll the iPod for certain event changes without enabling asynchronous remote event notification.

**Note:** Accessories must not poll for play status; they must use the iAP notification mechanism instead. See "[Command 0x08: SetRemoteEventNotification](#)" (page 185).

**Table 3-57** `GetRemoteEventStatus` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo

Byte number	Value	Comment
4	0x0A	Command ID: GetRemoteEventStatus
5	0xF1	Checksum

## Command 0x0B: RetRemoteEventStatus

Direction: iPod to Device

The iPod sends this command in response to ["Command 0x0A: GetRemoteEventStatus"](#) (page 193).

**Table 3-58** RetRemoteEventStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0B	Command ID: RetRemoteEventStatus
5	0xNN	remEventStatus (bits 31:24). The event status that has changed. See <a href="#">Table 3-49</a> (page 186) for a list of possible events.
6	0xNN	remEventStatus (bits 23:16)
7	0xNN	remEventStatus (bits 15:8)
8	0xNN	remEventStatus (bits 7:0)
9	0xNN	Checksum

The bits in [Table 3-49](#) (page 186) represent the events whose status has changed on the iPod since the last GetRemoteEventStatus command was received. For example, if the returned remEventStatus field has bits 2 and 7 set, then the chapter index and shuffle states of the iPod have changed since the last GetRemoteEventStatus command was sent by the accessory. Accessories can use ["Command 0x0C: GetiPodStateInfo"](#) (page 194) to get the updated state information for those events.

## Command 0x0C: GetiPodStateInfo

Direction: Device to iPod

The device obtains iPod state information. The information type (infoType) field specifies the type of information to get. In response, the iPod sends ["Command 0x0D: RetiPodStateInfo"](#) (page 196) with the requested state information.

**Table 3-59** GetiPodStateInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0C	Command ID: GetiPodStateInfo
5	0xNN	infoType (1 byte). The type of state information for which to query the iPod. See <a href="#">Table 3-60</a> (page 195).
6	0xNN	Checksum

[Table 3-60](#) (page 195) lists the different types of information for which you can query the iPod.

**Table 3-60** infoType values

Value	Type of information
0x00	Track time position in milliseconds
0x01	Track playback index
0x02	Chapter information
0x03	Play status (play, pause, stop, FF, and RW)
0x04	Mute and volume information
0x05	Power and battery status
0x06	Equalizer setting
0x07	Shuffle setting
0x08	Repeat setting
0x09	Date and time
0x0A	Alarm state and time
0x0B	Backlight state
0x0C	Hold switch state
0x0E	Audiobook speed
0x0F	Track time position in seconds

Value	Type of information
0x10	Mute/UI/Absolute volume
0x11–0xFF	Reserved

For example, to retrieve the iPod Equalizer Setting, an accessory could send the command shown in [Table 3-61](#) (page 196).

**Table 3-61** GetiPodStateInfo packet to retrieve the iPod Equalizer Setting

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0C	Command ID: GetiPodStateInfo
5	0x06	infoType = 0x06 (Equalizer setting)
6	0xE8	Checksum

## Command 0x0D: RetiPodStateInfo

Direction: iPod to Device

The iPod sends this command in response to "[Command 0x0C: GetiPodStateInfo](#)" (page 194). The format of the returned state information depends on the type of information. See [Table 3-51](#) (page 188) for a description of the data returned for each information type.

**Table 3-62** RetiPodStateInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0D	Command ID: RetiPodStateInfo
5	0xNN	infoType (1 byte). The type of iPod state information returned.
6 ... N	0xNN	infoData (variable length). The iPod state information.



Byte number	Value	Comment
(last byte)	0xNN	Checksum

For example, an accessory requesting the Chapter Info of the currently playing track would receive a `RetiPodStateInfo` command packet similar to this (assuming a track index of 10, a chapter count of 8 and a current chapter index of 3):

**Table 3-63** `RetiPodStateInfo` packet for requesting chapter information

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0B	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0D	Command ID: <code>RetiPodStateInfo</code>
5	0x02	<code>infoType</code> = 0x02 (Chapter Information)
6	0x00	<code>infoData</code> = Track Index (bits 31:24)
7	0x00	<code>infoData</code> = Track Index (bits 23:16)
8	0x00	<code>infoData</code> = Track Index (bits 15:8)
9	0x0A	<code>infoData</code> = Track Index (bits 7:0)
10	0x00	<code>infoData</code> = Chapter Count (bits 15:8)
11	0x08	<code>infoData</code> = Chapter Count (bits 7:0)
12	0x00	<code>infoData</code> = Chapter Index (bits 15:8)
13	0x03	<code>infoData</code> = Chapter Index (bits 7:0)
14	0xCE	Checksum

## Command 0x0E: `SetiPodStateInfo`

Direction: Device to iPod

Sets the iPod state. The information type (`infoType`) field specifies the type of information to update. In response, the iPod sends an ACK command with the results of the operation. Some commands include a `bRestoreOnExit` parameter that optionally allows the original iPod setting to be restored on exit.

**Table 3-64** SetiPodStateInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0E	Command ID: SetiPodStateInfo
5	0xNN	infoType (1 byte). The type of iPod state information to set.
6 ... N	0xNN	infoData (variable length). The data for the iPod state information to set.
(last byte)	0xNN	Checksum

Table 3-65 (page 198) lists the possible values of the `infoType` field and the corresponding data in the `infoData` field.

**Note:** Information types 0x09 (date/time) and 0x0B (backlight) cannot be set on the iPhone or iPod touch.

**Table 3-65** iPod state data

Information type		Information data	
Value	Name	Description	Length (bytes)
0x00	Track position	The new position of the track, in milliseconds. <ul style="list-style-type: none"> <li>Byte 0: Track Position (bits 31:24)</li> <li>Byte 1: Track Position (bits 23:16)</li> <li>Byte 2: Track Position (bits 15:8)</li> <li>Byte 3: Track Position (bits 7:0)</li> </ul>	0x04
0x01	Track index	The index of the track to play. <ul style="list-style-type: none"> <li>Byte 0: Track Index (bits 31:24)</li> <li>Byte 1: Track Index (bits 23:16)</li> <li>Byte 2: Track Index (bits 15:8)</li> <li>Byte 3: Track Index (bits 7:0)</li> </ul>	0x04
0x02	Chapter index	The new chapter index. <ul style="list-style-type: none"> <li>Byte 0: Chapter Index (bits 15:8)</li> <li>Byte 1: Chapter Index (bits 7:0)</li> </ul>	0x02

Information type		Information data	
Value	Name	Description	Length (bytes)
0x03	Play status	The play status of the iPod (play, pause, stop, FF or REW). See <a href="#">Table 3-52</a> (page 191) for a list of possible values. Byte 0: Play Status (bits 7:0)	0x01
0x04	Mute/volume	The new mute setting or volume level. <ul style="list-style-type: none"> <li>Byte 0: Mute State (bits 7:0) A value of 0x00 turns mute off; a value of 0x01 turns on mute.</li> <li>Byte 1: Volume Level (bits 7:0) A value between 0 and 255, with 0 indicating minimum volume and 255 indicating maximum volume.</li> <li>Byte 2: <code>bRestoreOnExit</code> (bits 7:0) See <a href="#">Table 3-66</a> (page 202).</li> </ul> If the mute state is 0x01, the volume level field is ignored.	0x03
0x05	Power/battery	Reserved: Power and battery state cannot be set.	N/A
0x06	Equalizer state	The new Equalizer Setting. <ul style="list-style-type: none"> <li>Byte 0: Equalizer index (bits 31:24)</li> <li>Byte 1: Equalizer index (bits 23:16)</li> <li>Byte 2: Equalizer index (bits 15:8)</li> <li>Byte 3: Equalizer index (bits 7:0)</li> <li>Byte 4: <code>bRestoreOnExit</code> (bits 7:0) See <a href="#">Table 3-66</a> (page 202).</li> </ul>	0x05
0x07	Shuffle	The new state of the shuffle setting. <ul style="list-style-type: none"> <li>Byte 0: Shuffle State (bits 7:0) See <a href="#">Table 3-53</a> (page 192) for a list of possible values.</li> <li>Byte 1: <code>bRestoreOnExit</code> (bits 7:0) See <a href="#">Table 3-66</a> (page 202).</li> </ul>	0x02
0x08	Repeat	The new state of the repeat setting <ul style="list-style-type: none"> <li>Byte 0: Repeat State (bits 7:0) See <a href="#">Table 3-54</a> (page 192) for a list of possible values.</li> <li>Byte 1: <code>bRestoreOnExit</code> (bits 7:0) See <a href="#">Table 3-66</a> (page 202).</li> </ul>	0x02

Information type		Information data	
Value	Name	Description	Length (bytes)
0x09	Date/time	<p>The new date and time.</p> <ul style="list-style-type: none"> <li>Bytes 0–1 specify the current year. A value of 2005 represents the year 2005 A.D. <ul style="list-style-type: none"> <li>Byte 0: Year (bits 15:8)</li> <li>Byte 1: Year (bits 7:0)</li> </ul> </li> <li>Byte 2: Month (bits 7:0) <ul style="list-style-type: none"> <li>A value between 1 and 12, where 1 = January and 12 = December.</li> </ul> </li> <li>Byte 3: Day of the month (bits 7:0) <ul style="list-style-type: none"> <li>A value between 1 and 31.</li> </ul> </li> <li>Byte 4: Hour (bits 7:0) <ul style="list-style-type: none"> <li>A value between 0 and 23, where 0 = 12:00 a.m. and 23 = 11:00 p.m.</li> </ul> </li> <li>Byte 5: Minute (bits 7:0) <ul style="list-style-type: none"> <li>A value between 0 and 59.</li> </ul> </li> </ul>	0x06
0x0A	Alarm	<p>The alarm state and time.</p> <ul style="list-style-type: none"> <li>Byte 0: Alarm State (bits 7:0) <ul style="list-style-type: none"> <li>A value of 0 turns the alarm off, while a value of 1 enables the alarm. When turning off the alarm, the Hour and Minutes bytes are ignored.</li> </ul> </li> <li>Byte 1: Alarm Hour (bits 7:0) <ul style="list-style-type: none"> <li>A value between 0 and 23, where 0 = 12:00 a.m. and 23 = 11:00 p.m.</li> </ul> </li> <li>Byte 2: Alarm Minute (bits 7:0) <ul style="list-style-type: none"> <li>A value between 0 and 59.</li> </ul> </li> <li>Byte 3: <code>bRestoreOnExit</code> (bits 7:0): See <a href="#">Table 3-66</a> (page 202).</li> </ul>	0x04
0x0B	Backlight	<p>The new state of the backlight.</p> <ul style="list-style-type: none"> <li>Byte 0: Backlight Level (bits 7:0) <ul style="list-style-type: none"> <li>The backlight control only supports two modes: on or off. A value of 0 turns the backlight off; any other value turns on the backlight. There is no support for dimming the iPod backlight setting.</li> </ul> </li> <li>Byte 1: <code>bRestoreOnExit</code> (bits 7:0) <ul style="list-style-type: none"> <li>See <a href="#">Table 3-66</a> (page 202).</li> </ul> </li> </ul>	0x02

Information type		Information data	
Value	Name	Description	Length (bytes)
0x0C	Hold switch	Reserved. The state of the Hold switch cannot be set.	N/A
0x0D	Sound check	<p>The new sound check state.</p> <ul style="list-style-type: none"> <li>Byte 0: Sound Check State (bits 7:0) A value of 0x00 turns sound check off; a value of 0x01 turns it on.</li> <li>Byte 1: <code>bRestoreOnExit</code> (bits 7:0) See <a href="#">Table 3-66</a> (page 202).</li> </ul>	0x02
0x0E	Audiobook speed	<p>The audiobook playback speed.</p> <ul style="list-style-type: none"> <li>Byte 0: Audiobook Playback Speed Setting (bits 7:0) See <a href="#">Table 3-56</a> (page 193) for a list of possible values.</li> <li>Byte 1: <code>bRestoreOnExit</code> (bits 7:0) See <a href="#">Table 3-66</a> (page 202).</li> </ul>	0x01
0x0F	Track position in seconds	<p>The current track time position, in seconds.</p> <ul style="list-style-type: none"> <li>Byte 0: Track Time Position (bits 15:8)</li> <li>Byte 1: Track Time Position (bits 7:0)</li> </ul>	0x02
0x10	Mute/UI/Absolute volume	<p>The current state of the mute setting, UI volume, and absolute volume.</p> <ul style="list-style-type: none"> <li>Byte 0: Mute state (bits 7:0). A value of 0 indicates that muting is off; a value of 1 indicates that muting is on.</li> <li>Byte 1: UI volume level (bits 7:0). A value between 0 and 255, normalized to volume limit settings. 0 indicates minimum volume and 255 indicates maximum volume. If the accessory sets this byte to 0, the iPod uses the absolute volume setting.</li> <li>Byte 2: Absolute volume level (bits 7:0). A value between 0 and 255, not normalized. 0 indicates minimum absolute volume and 255 indicates maximum absolute volume. If muting is on (byte 0 = 0x00), the absolute volume level is not valid and is returned as 0.</li> <li>Byte 3: <code>bRestoreOnExit</code> (bits 7:0). See <a href="#">Table 3-66</a> (page 202) for a list of values.</li> </ul>	0x04
0x11–0xFF	Reserved	N/A	N/A

[Table 3-66](#) (page 202) lists the possible values for the `bRestoreOnExit` parameter.

**Table 3-66** Restore-on-exit values

Value	Meaning
0x00	Do not save the original state.
0x01	Save the original state and restore it on exit.

For example, an accessory could send the command shown in [Table 3-67](#) (page 202) to set the iPod alarm to 8:15 a.m. and have the setting restored upon accessory detach.

**Table 3-67** SetiPodStateInfo packet to set the alarm

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0E	Command ID: SetiPodStateInfo
5	0x0A	infoType = 0x0A (Alarm)
6	0x01	infoData = Alarm State set to On (0x01).
7	0x08	infoData = Alarm Hour set to 8.
8	0x0F	infoData = Alarm Minute set to 15.
9	0x01	InfoData = bRestoreOnExit set to true (0x01).
10	0xC5	Checksum

As another example, an accessory could send the command shown in [Table 3-68](#) (page 202) to set the currently playing track on the iPod to track 1000.

**Table 3-68** SetiPodStateInfo packet for setting the current track

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0E	Command ID: SetiPodStateInfo

Byte number	Value	Comment
5	0x01	infoType = 0x01 (Track Index)
6	0x00	infoData = Track Index (bits 31:24). Sets the playback track index to 1000.
7	0x00	infoData = Track Index (bits 23:16)
8	0x03	infoData = Track Index (bits 15:8)
9	0xE8	infoData = Track Index (bits 7:0)
10	0xFC	Checksum

**Note:** SetiPodStateInfo commands that use information types that have data fields larger than one byte must be sure to send the data in big-endian format. See the example in [Table 3-68](#) (page 202).

## Command 0x0F: GetPlayStatus

Direction: Device to iPod

The device requests the current iPod play status information. In response, the iPod sends "[Command 0x10: RetPlayStatus](#)" (page 203) with the current play state, track index, track position, and track length.

**Table 3-69** GetPlayStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x0F	Command ID: GetPlayStatus
5	0xEC	Checksum

## Command 0x10: RetPlayStatus

Direction: iPod to Device

The iPod sends this command in response to "[Command 0x0F: GetPlayStatus](#)" (page 203) and returns the current iPod play status information. If the iPod is in a playing or paused state, the track index (`trackIndex`), track length (`trackTotMs`), and track position (`trackPosMs`) fields are valid. Otherwise, they should be ignored.

**Table 3-70** RetPlayStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0F	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x10	Command ID: RetPlayStatus
5	0xNN	playState (1 byte). The iPod playback engine state. See <a href="#">Table 3-52</a> (page 191) for a list of possible values. If the value of this field is 0x00, all of the subsequent track fields are invalid. The value 0x05 is reserved.
6	0xNN	trackIndex (bits 31:24). Specifies the index of the currently playing track.
7	0xNN	trackIndex (bits 23:16)
8	0xNN	trackIndex (bits 15:8)
9	0xNN	trackIndex (bits 7:0)
10	0xNN	trackTotMs (bits 31:24). Specifies the total length of the track, in milliseconds.
11	0xNN	trackTotMs (bits 23:16)
12	0xNN	trackTotMs (bits 15:8)
13	0xNN	trackTotMs (bits 7:0)
14	0xNN	trackPosMs (bits 31:24). The current position of the track, in milliseconds.
15	0xNN	trackPosMs (bits 23:16)
16	0xNN	trackPosMs (bits 15:8)
17	0xNN	trackPosMs (bits 7:0)
18	0xNN	Checksum

## Command 0x11: SetCurrentPlayingTrack

Direction: Device to iPod

The device sets the iPod's currently playing track to the track at the specified index. The total number of playing tracks can be obtained by sending "[Command 0x14: GetNumPlayingTracks](#)" (page 208). The playing track index is zero based, so the valid range is from 0x0 to numPlayTracks-1 (one less than the total count).



**Table 3-71** SetCurrentPlayingTrack packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x11	Command ID: SetCurrentPlayingTrack
5	0xNN	trackIndex (bits 31:24). The track index to begin playing.
6	0xNN	trackIndex (bits 23:16)
7	0xNN	trackIndex (bits 15:8)
8	0xNN	trackIndex (bits 7:0)
9	0xNN	Checksum

## Command 0x12: GetIndexedPlayingTrackInfo

Direction: Device to iPod

The device requests track information for the specified playing track index. The `infoType` field specifies the type of information to be returned, such as track title, artist name, album name, track genre, and track chapter information. In response, the iPod sends "[Command 0x13: RetIndexedPlayingTrackInfo](#)" (page 206) with the requested track information.

**Table 3-72** GetIndexedPlayingTrackInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x09	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x12	Command ID: GetIndexedPlayingTrackInfo
5	0xNN	infoType (1 byte). The type of track information to retrieve. See <a href="#">Table 3-74</a> (page 207) for a list of the available types of information.
6	0xNN	trackIndex (bits 31:24). The index of the track for which to retrieve information.
7	0xNN	trackIndex (bits 23:16)

Byte number	Value	Comment
8	0xNN	trackIndex (bits 15:8)
9	0xNN	trackIndex (bits 7:0)
10	0xNN	chapIndex (bits 15:8). The index of the chapter for which to retrieve information. This field is valid only when infoType is chapter information (0x01) and the track at trackIndex has chapters. Set the chapIndex fields to 0x00 if infoType is not chapter information and trackIndex does not have chapters.
11	0xNN	chapIndex (bits 7:0)
12	0xNN	Checksum

## Command 0x13: RetIndexedPlayingTrackInfo

Direction: iPod to Device

The iPod sends this command in response to "[Command 0x12: GetIndexedPlayingTrackInfo](#)" (page 205). It returns the requested type of information and data for the specified playing track. Data returned as strings are encoded as null-terminated UTF-8 character arrays. If the track information string does not exist, an empty null-terminated string is returned.

**Table 3-73** RetIndexedPlayingTrackInfo packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x13	Command ID: RetIndexedPlayingTrackInfo
5	0xNN	infoType. The type of track information being returned. See <a href="#">Table 3-74</a> (page 207) for possible values.
6 ... N	0xNN	infoData (variable length). The track information data.
(last byte)	0xNN	Checksum

[Table 3-74](#) (page 207) shows the data returned for each type of track information.

**Table 3-74** Track information data

Info type	Data type	Track information data	Data length
0x00	Track caps/info	<p>Track capabilities and information. This contains 3 fields:</p> <ul style="list-style-type: none"> <li>■ Bytes 0–3: Track Caps bitfields. Specifies the capabilities of the track. These bits have the following meanings:            Bit 0: If equal to 1, the track is an audiobook.            Bit 1: If equal to 1, the track has chapters.            Bit 2: Set to 1 if album artwork is available, 0 otherwise            Bit 3: If equal to 1, the track has song lyrics.            Bit 6:4: Reserved            Bit 7: Track contains video (a video podcast, music video, movie, or TV show)            Bit 8: Track is currently queued to play as a video            Bit 31:9: Reserved</li> <li>■ Bytes 4–7: TrackTotMs. The total length of the track in milliseconds.</li> <li>■ Bytes 8–9: ChapCount. If the track has chapters, the chapter count.</li> </ul>	10
0x01	Chapter time/name	<p>Chapter time and name, having two fields:</p> <ul style="list-style-type: none"> <li>■ Bytes 0–3: Chapter Time The chapter time in milliseconds. A value of 0 indicates there are no chapters.</li> <li>■ (variable length): Chapter Name The chapter name, as a UTF-8 string.</li> </ul>	4 + Variable
0x02	Artist name	The artist name, as a UTF-8 string.	Variable
0x03	Album name	The album name, as a UTF-8 string.	Variable
0x04	Genre name	The genre name, as a UTF-8 string.	Variable
0x05	Track title	The title of the track, as a UTF-8 string.	Variable
0x06	Composer name	The composer name, as a UTF-8 string.	Variable

Info type	Data type	Track information data	Data length
0x07	Lyrics	Track lyrics data, consisting of 3 fields: <ul style="list-style-type: none"> <li>■ Byte 0: Packet information bits. If set, these bits have the following meanings:               <ul style="list-style-type: none"> <li>Bit 0: If equal to 1, indicates that this is one of multiple packets.</li> <li>Bit 1: If equal to 1, this is the last packet (applicable only if bit 0 is equal to 1).</li> <li>Bits 7:2: Reserved; set to 0.</li> </ul> </li> <li>■ Bytes 1–2: Packet index (16-bit big-endian format)</li> <li>■ Bytes 3–NN: Track lyrics as a UTF-8 string (see <b>Note</b> below).</li> </ul>	Variable
0x08	Artwork count	Artwork count data. The artwork count is a sequence of 4-byte records; each record consists of a 2-byte <code>formatID</code> value followed by a 2-byte count of images in that format for this track. If the track contains no format IDs, this info type will return only 1 byte containing 0x08 and no records. For information about <code>formatID</code> values, see <a href="#">"Command 0x17: RetArtworkFormats"</a> (page 210).	Variable
0x09–0xFF	Reserved	N/A	N/A

**Note:** Track lyrics are formatted as a single null-terminated UTF8 string. If the lyrics string is too long to be carried within a single packet, then the string is broken into sections and carried within separate packets with distinct values for each section index. The last lyrics packet section contains the null terminator character for the full track lyrics string. Lyric sections before the last section do not include null terminators, and individual sections may not necessarily be valid UTF8 strings. Accessories must use the packet payload length to determine the length of each string section and assemble the full lyrics string by concatenating the individual substrings in order.

If `GetIndexedPlayingTrackInfo` specified a track not currently playing, a null string is returned. Lyrics for a track being played may take up to 5 seconds to return. A device may try to get the current lyrics every 0.5 seconds until either 5 seconds has elapsed or the track lyrics string has been returned.

## Command 0x14: GetNumPlayingTracks

Direction: Device to iPod

The device requests the total number of tracks playing in the iPod playback engine. The count can be used to select a different playing track or obtain information for a specific track. In response, the iPod sends ["Command 0x15: RetNumPlayingTracks"](#) (page 209).

**Table 3-75** GetNumPlayingTracks packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x14	Command = 0x14 (GetNumPlayingTracks)
5	0xE7	Checksum

## Command 0x15: RetNumPlayingTracks

---

Direction: iPod to Device

The iPod sends this command in response to ["Command 0x14: GetNumPlayingTracks"](#) (page 208) received from the device. It returns the total number of tracks queued in the playback engine.

**Table 3-76** RetNumPlayingTracks packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x15	Command ID: RetNumPlayingTracks
5	0xNN	numPlayTracks (bits 31:24). The total number of queued playing tracks.
6	0xNN	numPlayTracks (bits 23:16)
7	0xNN	numPlayTracks (bits 15:8)
8	0xNN	numPlayTracks (bits 7:0)
9	0xNN	Checksum

## Command 0x16: GetArtworkFormats

---

Direction: Device to iPod

The device sends this command to obtain the list of supported artwork formats on the iPod. No parameters are sent. See ["Transferring Album Art"](#) (page 179).

**Table 3-77** GetArtworkFormats packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x16	Command ID: GetArtworkFormats
5	0xE5	Checksum

## Command 0x17: RetArtworkFormats

Direction: iPod to Device

The iPod sends this command to the device in response to ["Command 0x16: GetArtworkFormats"](#) (page 209). Each format is described in a 7-byte record (formatID:2, pixelFormat:1, width:2, height:2). The formatID is used when sending GetTrackArtworkTimes. The device may return zero records if the iPod does not contain any artwork. See ["Transferring Album Art"](#) (page 179).

**Table 3-78** RetArtworkFormats packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x17	Command ID: RetArtworkFormats
NN	0xNN	formatID (bits 15:8) iPod-assigned value for this format
NN	0xNN	formatID (bits 7:0)
NN	0xNN	pixelFormat. See <a href="#">Table 3-79</a> (page 211).
NN	0xNN	imageWidth (bits 15:8). Number of pixels wide for each image.
NN	0xNN	imageWidth (bits 7:0)
NN	0xNN	imageHeight (bits 15:8). Number of pixels high for each image.

Byte number	Value	Comment
<i>NN</i>	<i>0xNN</i>	imageHeight (bits 7:0)
Previous 7 bytes may be repeated <i>NN</i> times		
<i>NN</i>	<i>0xNN</i>	Checksum

**Table 3-79** Display pixel format codes

Display pixel format	Code
Reserved	0x00
Monochrome, 2 bits per pixel	0x01
RGB 565 color, little-endian, 16 bpp	0x02
RGB 565 color, big-endian, 16 bpp	0x03
Reserved	0x04–0xFF

## Command 0x18: GetTrackArtworkData

---

Direction: Device to iPod

The device sends this command to the iPod to request the artwork image bitmap data for a given `trackIndex`, `formatID`, and `artworkIndex`. See ["Transferring Album Art"](#) (page 179).

**Table 3-80** GetTrackArtworkData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0C	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x18	Command ID: GetTrackArtworkData
5	<i>0xNN</i>	<code>trackIndex</code> (bits 31:24).
6	<i>0xNN</i>	<code>trackIndex</code> (bits 23:16)
7	<i>0xNN</i>	<code>trackIndex</code> (bits 15:8)
8	<i>0xNN</i>	<code>trackIndex</code> (bits 7:0)
9	<i>0xNN</i>	<code>formatID</code> (bits 15:8)

Byte number	Value	Comment
10	0xNN	formatID (bits 7:0)
11	0xNN	Time offset in milliseconds (bits 31:24)
12	0xNN	Time offset in milliseconds (bits 23:16)
13	0xNN	Time offset in milliseconds (bits 15:8)
14	0xNN	Time offset in milliseconds (bits 7:0)
15	0xNN	Checksum

## Command 0x19: RetTrackArtworkData

Direction: iPod to Device

The iPod sends this command in response to a ["Command 0x18: GetTrackArtworkData"](#) (page 211) command received from a device. Multiple `RetTrackArtworkData` commands may be necessary to transfer all the data because it will be too much to fit into a single packet. See ["Transferring Album Art"](#) (page 179).

This command returns the overall image width and height in pixels, the image row size in bytes, and the inset rectangle that contains the actual artwork content. The inset rectangle coordinates consist of two *x,y* pairs. Each *x* or *y* value is 2 bytes, so the total size of the inset rectangle coordinate set is 8 bytes.

The total image size in bytes is given by multiplying the row size by the image height. Padding may be inserted at the top, bottom, left, or right of the artwork to fit it to the iPod screen.

**Table 3-81** `RetTrackArtworkData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x03	Lingo ID: Display Remote lingo
4	0x19	Command ID: <code>RetTrackArtworkData</code>
5	0xNN	Descriptor packet index (15:8). These fields uniquely identify each packet in the <code>RetTrackArtworkData</code> transaction. The first packet is the descriptor packet, which always starts with an index of 0x0000.
6	0xNN	Descriptor packet index (7:0)
7	0xNN	Display pixel format code. See <a href="#">Table 3-79</a> (page 211).
8	0xNN	Image width in pixels (15:8)



Byte number	Value	Comment
9	0xNN	Image width in pixels (7:0)
10	0xNN	Image height in pixels (15:8)
11	0xNN	Image height in pixels (7:0)
12	0xNN	Inset rectangle, top-left point, x value (15:8)
13	0xNN	Inset rectangle, top-left point, x value (7:0)
14	0xNN	Inset rectangle, top-left point, y value (15:8)
15	0xNN	Inset rectangle, top-left point, y value (7:0)
16	0xNN	Inset rectangle, bottom-right point, x value (15:8)
17	0xNN	Inset rectangle, bottom-right point, x value (7:0)
18	0xNN	Inset rectangle, bottom-right point, y value (15:8)
19	0xNN	Inset rectangle, bottom-right point, y value (7:0)
20	0xNN	Row size in bytes (bits 31:24)
21	0xNN	Row size in bytes (bits 23:16)
22	0xNN	Row size in bytes (bits 15:8)
23	0xNN	Row size in bytes (bits 7:0)
24	0xNN...	Image pixel data (variable length)
NN	0xNN	Checksum

**Note:** In subsequent packets in the sequence (packets with a descriptor packet index greater than 0x0000), bytes 7 through 23 are omitted.

## Command 0x1A: GetPowerBatteryState

Direction: Device to iPod

The device sends this command to obtain the power and battery level state of the iPod. In response, the iPod sends "[Command 0x1B: RetPowerBatteryState](#)" (page 214) with the power and battery information.

**Table 3-82** GetPowerBatteryState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1A	Command ID: GetPowerBatteryState
5	0xE1	Checksum

## Command 0x1B: RetPowerBatteryState

Direction: iPod to Device

The iPod sends this command in response to "[Command 0x1A: GetPowerBatteryState](#)" (page 213), returning the current iPod power state and battery level.

**Note:** If an external power status (indicated by the value of the `powerStat` field) is returned, the battery level is invalid and is 0 on return.

**Table 3-83** RetPowerBatteryState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1B	Command ID: RetPowerBatteryState
5	0xNN	<code>powerStat</code> (1 byte). The battery power state. See <a href="#">Table 3-55</a> (page 192).
6	0xNN	<code>battLevel</code> (1 byte). The battery power level. This is a value between 00 (the battery is fully discharged, no power remains) and 255 (the battery is fully charged). This field is valid only if <code>powerStat</code> is Internal battery (0x00 or 0x01)
7	0xNN	Checksum

## Command 0x1C: GetSoundCheckState

Direction: Device to iPod

The device requests the iPod's current sound check setting. When enabled, sound check adjusts track playback volume to the same level. In response, the iPod sends "[Command 0x1D: RetSoundCheckState](#)" (page 215) with the current sound check state.

**Table 3-84** GetSoundCheckState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1C	Command ID: GetSoundCheckState
5	0xDF	Checksum

## Command 0x1D: RetSoundCheckState

---

Direction: iPod to Device

The iPod sends this command in response to "[Command 0x1C: GetSoundCheckState](#)" (page 214) and returns the current state of the sound check setting.

**Table 3-85** RetSoundCheckState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1D	Command ID: RetSoundCheckState
5	0xNN	sndChkState (1 byte). The current state of the sound check setting. A value of 0x00 indicates that sound check is off; 0x01 indicates that it is on.
6	0xNN	Checksum

## Command 0x1E: SetSoundCheckState

---

Direction: Device to iPod

The device sets the state of the iPod's sound check setting and optionally saves the previous sound check state to be restored on device detach. In response to this command, the iPod sends an ACK packet with the status of the command.

**Table 3-86** SetSoundCheckState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x03	Lingo ID: Display Remote lingo
4	0x1E	Command ID: SetSoundCheckState
5	0xNN	sndChkState (1 byte). The new state of the sound check setting. A value of 0x00 turns off sound check off; 0x01 turns on sound check.
6	0xNN	bRestoreOnExit (1 byte). Specifies whether the original sound check state is saved and restored on exit. See <a href="#">Table 3-66</a> (page 202)
7	0xNN	Checksum

## Command 0x1F: GetTrackArtworkTimes

Direction: Device to iPod

The device sends this command to the iPod to request the list of artwork time offsets for a track. A 4-byte `trackIndex` specifies which track is to be selected. A 2-byte `formatID` indicates which type of artwork is desired. The format IDs that the iPod supports can be obtained using the "[Command 0x16: GetArtworkFormats](#)" (page 209) command.

The 2-byte `artworkIndex` specifies where to begin searching for artwork. A value of 0 indicates that the iPod should start with the first available artwork.

The 2-byte `artworkCount` specifies the maximum number of times to be returned. A value of -1 (0xFFFF) indicates that all artwork times from the specified index should be returned.

**Table 3-87** GetTrackArtworkTimes packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0C	Length of packet
3	0x03	Lingo ID: Display Remote lingo

Byte number	Value	Comment
4	0x1F	Command ID: GetTrackArtworkTimes
5	0xNN	trackIndex (bits 31:24)
6	0xNN	trackIndex (bits 23:16)
7	0xNN	trackIndex (bits 15:8)
8	0xNN	trackIndex (bits 7:0)
9	0xNN	formatID (bits 15:8)
10	0xNN	formatID (bits 7:0)
11	0xNN	artworkIndex (bits 15:8)
12	0xNN	artworkIndex (bits 7:0)
13	0xNN	artworkCount (bits 15:8)
14	0xNN	artworkCount (bits 7:0)
15	0xNN	Checksum

## Command 0x20: RetTrackArtworkTimes

Direction: iPod to Device

The iPod sends this command to the device in response to a "[Command 0x1F: GetTrackArtworkTimes](#)" (page 216) command. The device returns zero or more 4-byte records, one for each piece of artwork associated with the track and format specified by `GetTrackArtworkTimes`. Artwork times are expressed as offsets, in milliseconds, from the beginning of the track.

The number of records returned will be no greater than the number specified in the `GetTrackArtworkTimes` command. It may, however, be less than requested. This can happen if there are fewer pieces of artwork available than were requested, or if the iPod is unable to place the full number in a single packet. Check the number of records returned against the results of `RetIndexedPlayingTrackInfo` with `infoType 0x08` to ensure that all artwork has been received.

**Table 3-88** RetTrackArtworkTimes packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x03	Lingo ID: Display Remote lingo

Byte number	Value	Comment
4	0x20	Command ID: RetTrackArtworkTimes
5	0xNN	Time offset in milliseconds (bits 31:24)
6	0xNN	Time offset in milliseconds (bits 23:16)
7	0xNN	Time offset in milliseconds (bits 15:8)
8	0xNN	Time offset in milliseconds (bits 7:0)
The preceding 4 bytes may be repeated NN times.		
NN	0xNN	Checksum

## Lingo 0x04: Extended Interface Lingo

Lingo 0x04 of the iAP is specified in detail in ["The Extended Interface Protocol"](#) (page 341).

## Lingo 0x05: Accessory Power Lingo

The Accessory Power lingo is used by devices that draw up to 100 mA peak current from the iPod through pin 13 of the 30-pin connector (Accessory Detect). To use this lingo, accessories must identify themselves as intermittent high-power devices during the identification process. They may include accessories that transmit the iPod analog audio over radio frequencies, typically over an unused frequency in the FM band.

The Accessory Power lingo is intended for use in conjunction with audio playback from the iPod. The accessory must remain in its low-power state until it receives a `BeginHighPower` command, which notifies it that it may begin drawing high power. The `EndHighPower` command notifies the accessory that it must stop drawing high power and return to the low power state within 1 second of receiving the command.

Like all accessories, an accessory using the Accessory Power lingo must comply with the power requirements of the iPod's Sleep and Hibernate states, as specified in ["iPod Power States and Accessory Power"](#) (page 341). Accessories are informed of iPod state changes by receiving a General lingo `NotifyiPodStateChange` command.

**Table 3-89** Accessory Power lingo command summary

Command	ID	Data length	Protocol version	Authentication required
Reserved	0x00–0x01	N/A	N/A	N/A
<code>BeginHighPower</code>	0x02	0x00	All	UART: No
<code>EndHighPower</code>	0x03	0x00	All	USB: Yes
Reserved	0x04–0xFF	N/A	N/A	N/A

## Command History of the Accessory Power lingo

Table 3-90 (page 219) shows the history of command changes in the Accessory Power lingo:

**Table 3-90** Accessory Power lingo command history

Lingo version	Command changes	Features
(0.00)	Add: 0x02–0x03	Begin/EndHighPower support
1.00	None	Version number available through RequestLingoProtocol - Version
1.01	None	BugFix: BeginTransmission command sent after device inserted while iPod is playing

## Command 0x02: BeginHighPower

Direction: iPod to Device

The iPod sends this command to notify the device that high power may be used. Upon receipt of the this command, the device may begin drawing more than 5 mA power, up to a maximum of 100 mA, if it has previously requested intermittent high power during its identification process.

**Table 3-91** BeginHighPower packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x05	Lingo ID: Accessory Power lingo
4	0x02	Command ID: BeginHighPower
5	0xF7	Checksum

## Command 0x03: EndHighPower

Direction: iPod to Device

The iPod sends this command to notify the device to stop using accessory high power. The device must reduce its power usage to less than 5 mA within 1 second of receiving an EndHighPower packet.

**Note:** Failure to reduce device power consumption to below 5 mA within 1 second after receiving this notification could damage the iPod.

**Table 3-92** EndHighPower packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x05	Lingo ID: Accessory Power lingo
4	0x03	Command ID: EndHighPower
5	0xF6	Checksum

## Lingo 0x07: RF Tuner Lingo

An external radio frequency tuner accessory can be attached to an iPod to provide radio reception. When an iPod successfully acknowledges an accessory's declaration of the RF Tuner lingo during the identification process, it makes a radio menu item available. Choosing this menu item displays the iPod tuner application. The application lets the user change the iPod's music source and control the accessory's RF band and tuner frequency.

**Note:** If an iPod contains an RF Tuner (such as the FM tuner in the 5G nano), the external tuner accessory will override it.

The RF Tuner lingo is used to pass control and state information between an iPod and an RF tuner device. Normally the iPod is the master and the accessory responds to commands from the iPod. This means that the iPod can initiate actions such as controlling tuner power, setting the tuner's band and frequency, initiating up or down frequency scans, and so on. An iPod attached to an RF tuner device also stores station frequencies and other tuner state information.

Every RF tuner device must report all of its capabilities back to the attached iPod, including support for at least one of the RF bands listed in [Table 3-100](#) (page 227). Based on the capabilities reported by the RF tuner device, the iPod tuner application may choose to change its appearance to reflect the presence or absence of certain RF tuner features.

## RF Tuner Accessory Design

RF tuner accessory devices for iPods must meet the following requirements:

- All RF tuner lingo commands require authentication.



- An iPod can support only one RF tuner device at any time, and that device must be attached using the 30-pin connector.
- On reset or power up, the RF tuner device must be in its low power state (<5 mA) with the tuner off and any audio output disabled.
- US devices must support the Europe/US FM band (87.5–108.0 MHz), with 200 KHz channel spacing and default 75 µsec deemphasis.
- EU/US devices must support the Europe/US FM band (87.5–108.0 MHz), with 100 KHz channel spacing and selectable 50 µsec or 75 µsec deemphasis.
- JP devices must support the Japan FM band (76.0–90.0 MHz), with 100 KHz channel spacing and selectable 50 µsec or 75 µsec deemphasis.

An RF tuner device may support US, EU, and/or JP requirements.

The following options apply to RF tuner devices:

- An RF tuner device may send asynchronous notification of state changes to an attached iPod, but such notifications are not required.
- RF tuner devices are not expected to retain state information across accessory power down cycles resulting from the invocation of Hibernate mode.
- RF tuner devices may support HD radio.

## RF Tuner Power

---

An attached RF tuner device will be notified of iPod state changes such as transitions between Power On, Sleep, and Hibernate states. It is responsible for keeping its power consumption below the maximum allowed limits for each iPod state. For details, see ["Accessory Power Policy"](#) (page 39).

Accessory power is completely shut off during hibernation. When an iPod wakes from hibernation and the accessory detects accessory power, the accessory must wait at least 80 ms and then start the IDPS process. When an iPod transitions from sleep to power on, it sends a `RequestIdentify` command to the accessory. In either case, the accessory must reidentify and reauthenticate itself, using IDPS (or `IdentifyDeviceLingoes` if the iPod does not support IDPS).

RF Tuner accessories can register for intermittent high power during the identification process. If properly registered, they may draw up to 100 mA after receiving a `SetTunerCtrl` command specifying power-on. They must reduce their power consumption below 5 mA within 1 second of receiving a `SetTunerCtrl` command specifying power-off. Regardless of the power state specified by the iPod, all devices must comply with the iPod state changes cited above.

## RF Tuner Lingo Commands

---

[Table 3-93](#) (page 222) summarizes the RF Tuner commands (lingo 0x07).

**Table 3-93** RF Tuner lingo command summary

Command	ID	Direction	Data length	Protocol version	Authentication required
ACK	0x00	Dev to iPod	4 or 8	1.00	Yes
GetTunerCaps	0x01	iPod to Dev	2	1.00	Yes
RetTunerCaps	0x02	Dev to iPod	8	1.00	Yes
GetTunerCtrl	0x03	iPod to Dev	2	1.00	Yes
RetTunerCtrl	0x04	Dev to iPod	3	1.00	Yes
SetTunerCtrl	0x05	iPod to Dev	3	1.00	Yes
GetTunerBand	0x06	iPod to Dev	2	1.00	Yes
RetTunerBand	0x07	Dev to iPod	3	1.00	Yes
SetTunerBand	0x08	iPod to Dev	3	1.00	Yes
GetTunerFreq	0x09	iPod to Dev	2	1.00	Yes
RetTunerFreq	0x0A	Dev to iPod	7	1.00	Yes
SetTunerFreq	0x0B	iPod to Dev	6	1.00	Yes
GetTunerMode	0x0C	iPod to Dev	2	1.00	Yes
RetTunerMode	0x0D	Dev to iPod	3	1.00	Yes
SetTunerMode	0x0E	iPod to Dev	3	1.00	Yes
GetTunerSeekRssi	0x0F	iPod to Dev	2	1.00	Yes
RetTunerSeekRssi	0x10	Dev to iPod	3	1.00	Yes
SetTunerSeekRssi	0x11	iPod to Dev	3	1.00	Yes
TunerSeekStart	0x12	iPod to Dev	3	1.00	Yes
TunerSeekDone	0x13	Dev to iPod	7	1.00	Yes
GetTunerStatus	0x14	iPod to Dev	2	1.00	Yes
RetTunerStatus	0x15	Dev to iPod	3	1.00	Yes
GetStatusNotifyMask	0x16	iPod to Dev	2	1.00	Yes
RetStatusNotifyMask	0x17	Dev to iPod	3	1.00	Yes
SetStatusNotifyMask	0x18	iPod to Dev	3	1.00	Yes
StatusChangeNotify	0x19	Dev to iPod	3	1.00	Yes

Command	ID	Direction	Data length	Protocol version	Authentication required
GetRdsReadyStatus	0x1A	iPod to Dev	2	1.00	Yes
RetRdsReadyStatus	0x1B	Dev to iPod	6	1.00	Yes
GetRdsData	0x1C	iPod to Dev	3	1.00	Yes
RetRdsData	0x1D	Dev to iPod	0xNN	1.00	Yes
GetRdsNotifyMask	0x1E	iPod to Dev	2	1.00	Yes
RetRdsNotifyMask	0x1F	Dev to iPod	6	1.00	Yes
SetRdsNotifyMask	0x20	iPod to Dev	6	1.00	Yes
RdsReadyNotify	0x21	Dev to iPod	0xNN	1.00	Yes
Reserved	0x22–0x24	N/A	N/A	N/A	N/A
GetHdProgramServiceCount	0x25	iPod to Dev	0	1.01	Yes
RetHdProgramServiceCount	0x26	Dev to iPod	1	1.01	Yes
GetHdProgramService	0x27	iPod to Dev	0	1.01	Yes
RetHdProgramService	0x28	Dev to iPod	1	1.01	Yes
SetHdProgramService	0x29	iPod to Dev	1	1.01	Yes
GetHddDataReadyStatus	0x2A	iPod to Dev	0	1.01	Yes
RetHddDataReadyStatus	0x2B	Dev to iPod	4	1.01	Yes
GetHddData	0x2C	iPod to Dev	1	1.01	Yes
RetHddData	0x2D	Dev to iPod	0xNN	1.01	Yes
GetHddDataNotifyMask	0x2E	iPod to Dev	0	1.01	Yes
RetHddDataNotifyMask	0x2F	Dev to iPod	4	1.01	Yes
SetHddDataNotifyMask	0x30	iPod to Dev	4	1.01	Yes
HddDataReadyNotify	0x31	Dev to iPod	0xNN	1.01	Yes
Reserved	0x32–0xFF	N/A	N/A	N/A	N/A

## Command History of the RF Tuner Lingo

Table 3-94 (page 224) shows the history of changes to the RF Tuner lingo.

**Table 3-94** RF Tuner lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x21	RF tuner control and support
1.01	Update: 0x02, 0x04, 0x05, 0x07, 0x0D, 0x0E, 0x12, 0x13, 0x15, 0x17, 0x18, 0x19, 0x1B, 0x1C, 0x1D, 0x1F, 0x20, and 0x21 Add: 0x25–0x31	Support for HD radio, AM tuning, FM wide band

## Command 0x00: ACK

Direction: Device to iPod

This command is sent by an RF tuner device on completion of a command received from an iPod. An ACK response is also sent if a bad parameter is received, an unsupported or invalid command is received, or a command that does not return any data has finished.

**Note:** Certain bytes of the RF tuner lingo ACK command packet are included only if the value of `cmdStatus` is 0x06. See [Table 3-95](#) (page 224) and [Table 3-96](#) (page 224).

**Table 3-95** RF tuner lingo ACK packet with `cmdStatus` not 0x06

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x00	Command ID: ACK
5	0xNN	<code>cmdStatus</code> : Status of command received (see <a href="#">Table 3-97</a> (page 225))
6	0xNN	<code>cmdIDOrig</code> : ID of the command for which the response is being sent
7	0xNN	Checksum

**Table 3-96** RF tuner lingo ACK packet with `cmdStatus` equal to 0x06

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x00	Command ID: ACK
5	0x06	cmdStatus: Command pending
6	0xNN	cmdIDOrig: ID of the command for which the response is being sent
7	0xNN	cmdPendTime (bits 31:24). Total timeout in milliseconds for the pending command to complete.
8	0xNN	cmdPendTime (bits 23:16)
9	0xNN	cmdPendTime (bits 15:8)
10	0xNN	cmdPendTime (bits 7:0)
11	0xNN	Checksum

**Table 3-97** RF tuner lingo ACK command status values

Value	Meaning
0x00	Command OK
0x01	Unknown track category (not applicable)
0x02	Command failed (command valid but did not succeed)
0x03	Out of resources (iPod internal allocation failed)
0x04	Bad parameter (command or input parameters invalid)
0x05	Unknown track ID (not applicable)
0x06	Command pending (cmdPendTime parameter returned)
0x07	Not authenticated (iPod not authenticated)

**Note:** A value of 0x06 is usually returned in the cmdStatus byte of the RF tuner ACK packet for commands that require more than 100 ms to complete. In this case, the cmdPendTime parameter is included in bytes 7–10 of the ACK packet. If the completion status is not returned by the total number of milliseconds specified by these bytes, the iPod will assume that a command failure has occurred and will retry the command or otherwise recover from the error. The RF tuner device should not return any response to the command after this timeout period has expired.

## Command 0x01: GetTunerCaps

Direction: iPod to Device

This command is sent by an iPod to query an attached RF tuner device's capabilities and determine what features the device supports. In response, the device must send a `RetTunerCaps` command with a payload specifying its capabilities.

**Table 3-98** GetTunerCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x01	Command ID: GetTunerCaps
5	0xF6	Checksum

## Command 0x02: RetTunerCaps

Direction: Device to iPod

This command is sent by an RF tuner device in response to a `GetTunerCaps` command sent by an iPod. The command transmits a payload indicating which RF tuner capabilities it supports.

**Table 3-99** RetTunerCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x02	Command ID: RetTunerCaps
5	0xNN	Tuner capabilities (bits 31:24) (See <a href="#">Table 3-100</a> (page 227))
6	0xNN	Tuner capabilities (bits 23:16)
7	0xNN	Tuner capabilities (bits 15:8)
8	0xNN	Tuner capabilities (bits 7:0)

Byte number	Value	Comment
9	0x01	Nonzero reserved; set to 0x01
10	0x00	Reserved; set to 0x00
11	0xNN	Checksum

**Note:** The capabilities bits returned in the `RetTunerCaps` payload (bytes 5-8) correspond to the band, control, mode, and status commands of the RF tuner lingo. The iPod will enable features or use commands only if the RF tuner device sets the associated capabilities bits when sending this command.

**Table 3-100** RF tuner device capabilities payload

Bits	Capability
00	AM band, Worldwide (520-1710 KHz) capable
01	FM band, Europe/US (87.5–108.0 MHz) capable
02	FM band, Japan (76.0–90.0 MHz) capable
03	FM band, Wide (76.0-108.0 MHz) capable
04	HD Radio capable; may be set only if bit 01 is set
07:05	Reserved; set to 0
08	Tuner power on/off control capable
09	Status change notification capable
15:10	Reserved; set to 0
17:16	Minimum FM resolution ID bits (see <a href="#">Table 3-101</a> (page 228))
18	Tuner seek up/down capable
19	Tuner seek RSSI threshold capable (must not be set if bit 18 is 0)
20	Force monophonic mode capable
21	Stereo blend capable
22	FM Tuner deemphasis select capable
23	AM tuner resolution 9KHz (0=10KHz only) capable
24	Radio Data System (RDS/RBDS) data capable
25	Tuner channel RSSI indication capable
26	Stereo source indicator capable

Bits	Capability
27	RDS/RDBS Raw mode capable
31:28	Reserved; set to 0

**Note:** Bit 04 (HD Radio capable), listed in [Table 3-100](#) (page 227), may be set in addition to the FM band bits (00-03), but only if bit 01 is set. This bit indicates that the radio is capable of receiving HD signals on an existing Europe/US AM or FM band.

**Table 3-101** Minimum FM resolution ID bits

ID bits	Description
00	200 kHz capable
01	100 kHz capable
10	50 kHz capable
11	Reserved

**Note:** The minimum FM resolution bits (bits 17-16) should report the smallest FM tuner resolution that the RF tuner device supports.

## Command 0x03: GetTunerCtrl

Direction: iPod to Device

An iPod sends this command to an RF tuner device to get the device's control state. In response, the device must send a `RetTunerCtrl` command with its current control state.

**Table 3-102** `GetTunerCtrl` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x03	Command ID: <code>GetTunerCtrl</code>
5	0xF4	Checksum



## Command 0x04: RetTunerCtrl

Direction: Device to iPod

An RF tuner device uses this command to send its current device control state in response to a `GetTunerCtrl` command from an iPod. Tuner control bits may be set only if the corresponding capabilities bits have been set in a previous `RetTunerCaps` command.

**Table 3-103** `RetTunerCtrl` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x04	Command ID: <code>RetTunerCtrl</code>
5	0xNN	Tuner control state (see <a href="#">Table 3-104</a> (page 229))
6	0xNN	Checksum

**Table 3-104** Tuner control state bits

Bit	Description
0	RF tuner device power is on (1) or off (0). When RF tuner device power is off, the accessory should rest in the lowest power state that still allows iAP commands to be received and processed.
1	Status change notification is enabled (1) or disabled (0). When status change notification is disabled, the iPod assumes that the device will send no asynchronous <code>StatusChangeNotify</code> commands. The iPod may poll the device for changes.
2	Reserved
3	RDS/RBDS Raw mode enabled
7:4	Reserved

## Command 0x05: SetTunerCtrl

Direction: iPod to Device

This command is sent by an iPod to control an RF tuner device's state. In response, the device must return an `ACK` command with the command status. RF tuner state information, such as tuner frequency, band, and so on, must be preserved by the device across tuner on and off cycles, assuming accessory power is available.

When the tuner power is turned off, it must disable its audio output and reduce its power consumption to less than 5 mA. When tuner power is switched on, its power consumption may rise to the maximum declared during its identification process. If a device has specified that it requires intermittent accessory high power (up to 100 mA while on), the iPod will enable its accessory high power before sending a `SetTunerCtrl` command to turn on tuner power. The iPod accessory high power will remain on until a subsequent `SetTunerCtrl` command to turn off tuner power has finished. The accessory must reduce its power consumption below 5 mA within 1 second of receiving the command.

**Note:** The iPod `NotifyiPodStateChange` command overrides this command for iPod transitions to a low power state (Sleep or Hibernate). Even if this command has enabled RF tuner device power, the device must reduce its power consumption below 5 mA on receiving an iPod state change notification that specifies a transition to a low power state. The only iPod power state in which an RF tuner device requesting high power can consume more than 5 mA is the Power On state.

**Table 3-105** `SetTunerCtrl` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x05	Command ID: <code>SetTunerCtrl</code>
5	0xNN	Tuner control bits (see <a href="#">Table 3-106</a> (page 230))
6	0xNN	Checksum

**Table 3-106** Tuner control bits

Bit	Description
0	Turn RF tuner device power on (1) or off (0). When RF tuner device power is turned off, the accessory should rest in the lowest power state that still allows iAP commands to be received and processed.
1	Enable (1) or disable (0) status change notification. When status change notification is disabled, the iPod assumes that the device will send no asynchronous <code>StatusChangeNotify</code> commands. The iPod may poll the device for changes.
2	Reserved
3	Enable RDS/RBDS Raw mode. When enabled, raw RDS/RBDS data format is used by the RDS data commands (commands 0x1A-0x21). When disabled, parsed RDS data is used by the RDS data commands.
7:4	Reserved

## Command 0x06: GetTunerBand

Direction: iPod to Device

This command is sent by an iPod to get an RF tuner device's RF band information. The device must respond by returning a `RetTunerBand` command.

**Table 3-107** `GetTunerBand` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x06	Command ID: <code>GetTunerBand</code>
5	0xF1	Checksum

## Command 0x07: RetTunerBand

Direction: Device to iPod

This command is sent by an RF tuner device to report its tuner band state in response to an iPod's `GetTunerBand` command. If the RF tuner device power is off, it should return its last active band state.

**Note:** Tuner band state information must be consistent with the device's capabilities, as previously reported by a `RetTunerCaps` command.

**Table 3-108** `RetTunerBand` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x07	Command ID: <code>RetTunerBand</code>
5	0xNN	Tuner band state information (see <a href="#">Table 3-109</a> (page 232)):
6	0xNN	Checksum

**Table 3-109** Tuner band state IDs

ID	Description
0x00	AM band worldwide (520-1710 KHz)
0x01	Europe/US FM band (87.5–108.0 MHz)
0x02	Japan FM band (76.0–90.0 MHz)
0x03	FM wide band (76.0-108.0 MHz)
0x04–0xFF	Reserved

## Command 0x08: SetTunerBand

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to set its tuner band. In response, the device must send the iPod an **ACK** command with the command status. The command status must be reported as command failed (command status 0x02) if RF tuner device power is not on.

**Table 3-110** SetTunerBand packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x08	Command ID: SetTunerBand
5	0xNN	Tuner band to be set (see <a href="#">Table 3-109</a> (page 232))
6	0xNN	Checksum

**Note:** The tuner band setting requested by the iPod will be consistent with the RF tuner device's capabilities, as previously reported by a **RetTunerCaps** command.

## Command 0x09: GetTunerFreq

Direction: iPod to Device

This command is sent by an iPod to get an RF tuner device's current device tuner frequency and signal strength level. In response, the device must send the iPod a **RetTunerFreq** command.

**Table 3-111** GetTunerFreq packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x09	Command ID: GetTunerFreq
5	0xEE	Checksum

## Command 0x0A: RetTunerFreq

---

Direction: Device to iPod

This command is sent by an RF tuner device in response to a GetTunerFreq or SetTunerFreq command received from an iPod. The tuner frequency is expressed in kilohertz: for example, 76000 for 76.0 MHz, 87500 for 87.5 MHz, or 107900 for 107.9 MHz. Valid ranges are 87500 to 107900 for the European/US FM band and 76000 to 89900 for the Japanese FM band.

If the RF tuner device's capabilities includes tuner channel RSSI indication, byte 9 of the command packet may be a number greater than zero; otherwise it must be set to 0x00. The tuner RSSI level must be normalized to a value between 0 (minimum) and 255 (maximum). If RF tuner device power is off, the last active tuner frequency and signal strength (if applicable) should be sent.

**Table 3-112** RetTunerFreq packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0A	Command ID: RetTunerFreq
5	0xNN	Tuner frequency in kilohertz (bits 31:24)
6	0xNN	Tuner frequency (bits 23:16)
7	0xNN	Tuner frequency (bits 15:8)
8	0xNN	Tuner frequency (bits 7:0)

Byte number	Value	Comment
9	0xNN	Tuner channel received signal strength ( <code>rssiLevel</code> ); may range from 0x00 (no signal present or RSSI indication not supported) to 0xFF (maximum RSSI signal strength).
10	0xNN	Checksum

**Note:** The tuner channel received signal strength value is valid only if the tuner RSSI bit was set in a previous `RetTunerCaps` command.

## Command 0x0B: SetTunerFreq

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to select a specific tuner frequency within the current tuner band. The tuner frequency is expressed in kilohertz; valid ranges are 87500 to 107900 for the European/US FM band and 76000 to 89900 for the Japanese FM band. In response, the device must send a `RetTunerFreq` command with the new tuner frequency and RSSI level (if the device supports RSSI). The requested frequency should be within the currently selected tuner band. The command status must be reported as command failed (command status 0x02) if RF tuner device power is not on.

**Table 3-113** SetTunerFreq packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0B	Command ID: SetTunerFreq
5	0xNN	Tuner frequency in kilohertz (bits 31:24)
6	0xNN	Tuner frequency (bits 23:16)
7	0xNN	Tuner frequency (bits 15:8)
8	0xNN	Tuner frequency (bits 7:0)
9	0xNN	Checksum

## Command 0x0C: GetTunerMode

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to get the current tuner mode state from the device. In response, the device must send the iPod a `RetTunerMode` command.

**Table 3-114** `GetTunerMode` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0C	Command ID: <code>GetTunerMode</code>
5	0xEB	Checksum

## Command 0x0D: `RetTunerMode`

Direction: Device to iPod

This command is sent by a device in response to a `GetTunerMode` command received from an iPod. The tuner mode bits returned are valid only if the associated mode bits returned by a previous `RetTunerCaps` command were set. If tuner power is off, the last active tuner mode information should be returned.

**Table 3-115** `RetTunerMode` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0D	Command ID: <code>RetTunerMode</code>
5	0xNN	Tuner mode status (see <a href="#">Table 3-116</a> (page 235))
6	0xNN	Checksum

**Table 3-116** RF Tuner mode status bits

Bits	Description
1:0	FM tuner resolution (see <a href="#">Table 3-101</a> (page 228))
2	Tuner is currently seeking up or down

Bits	Description
3	Tuner is currently seeking with an RSSI minimum threshold enabled
4	Monophonic mode is forced (1) or stereo is allowed (0)
5	Stereo blend is enabled (valid only if bit 4 is 0)
6	FM Tuner deemphasis is 50 $\mu$ sec (1) or 75 $\mu$ sec (0)
7	AM tuner resolution (1 = 9 KHz, 0 = 10 KHz)

## Command 0x0E: SetTunerMode

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to set the current tuner mode. In response, the device must send an RF tuner lingo *ACK* command with the command status. The command status must be reported as command failed (command status 0x02) if RF tuner device power is not on.

**Table 3-117** SetTunerMode packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0E	Command ID: SetTunerMode
5	0xNN	Tuner mode to be set (see <a href="#">Table 3-118</a> (page 236))
6	0xNN	Checksum

**Table 3-118** Set RF Tuner mode bits

Bits	Description
1:0	Set FM tuner resolution (see <a href="#">Table 3-101</a> (page 228))
2-3	Reserved; set to 0
4	Force monophonic mode (1) or allow stereo (0)
5	Enable stereo blend (valid only if bit 4 is 0); this blends the source left and right channels to improve the perceived signal quality while still retaining some stereo image separation
6	Set FM Tuner deemphasis to 50 $\mu$ sec (1) or 75 $\mu$ sec (0)



Bits	Description
7	Set AM tuner resolution (1 = 9 KHz, 0 = 10 KHz)

**Note:** `SetTunerMode` must not be used to start tuner seeking or set the seek type. The `TunerSeekStart` command must be used instead.

## Command 0x0F: GetTunerSeekRssi

Direction: iPod to Device

This command is sent by an iPod to get the current tuner seek threshold value from an RF tuner device. In response, the device must send a `RetTunerSeekRssi` command with the seek RSSI threshold, if supported by the device. If seek RSSI is not supported, the RF tuner device must return an `ACK` command with failure status.

**Table 3-119** `GetTunerSeekRssi` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x0F	Command ID: <code>GetTunerSeekRssi</code>
5	0xE8	Checksum

## Command 0x10: RetTunerSeekRssi

Direction: Device to iPod

This command is sent by a device in response to a `GetTunerSeekRssi` command received from an iPod, assuming the device supports the seek RSSI capability. Its threshold value represents the minimum signal strength that allows a tuner channel to be recognized during the seek process; it is normalized to a value between 0 (minimum) and 255 (maximum). If device power is off, the last active RSSI threshold value should be sent.

**Table 3-120** `RetTunerSeekRssi` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x10	Command ID: <code>RetTunerSeekRssi</code>
5	0xNN	RSSI threshold for seeking action
6	0xNN	Checksum

## Command 0x11: `SetTunerSeekRssi`

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device that supports the seek RSSI capability, to set the device's seek RSSI signal strength threshold. The threshold value is the minimum signal strength that allows a tuner channel to be recognized during the seek process; it is normalized to a range between 0 (minimum) and 255 (maximum).

**Table 3-121** `SetTunerSeekRssi` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x11	Command ID: <code>SetTunerSeekRssi</code>
5	0xNN	RSSI threshold for seeking action
6	0xNN	Checksum

## Command 0x12: `TunerSeekStart`

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device that supports the tuner seek up/down capability (as reported by a previous `RetTunerCaps` command), to initiate seeking of a specified type. The returned `ACK` command status must be reported as command failed (command status 0x02) if RF tuner device power is not on. Seeking operations that use an RSSI threshold are initiated only if the RF tuner device's seek RSSI threshold capability is also supported, as reported by the `RetTunerCaps` command.

**Table 3-122** TunerSeekStart packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x12	Command ID: TunerSeekStart
5	0xNN	ID of tuner seeking operation (see <a href="#">Table 3-123</a> (page 239) and Note below)
6	0xNN	Checksum

**Note:** An accessory that declares support for HD radio by setting its capability bit 04 (see [Table 3-100](#) (page 227)) must also support HD seeks. If the accessory receives a seek code it does not support, it must return a DevACK command with Bad Parameter status (0x04).

**Table 3-123** Tuner seeking operations

ID	Operation	Use RSSI threshold
0x00	No seek operation (cancel seek operation, if active)	N/A
0x01	Seek up from beginning of band	No
0x02	Seek down from end of band	No
0x03	Seek up from current frequency	No
0x04	Seek down from current frequency	No
0x05	Seek up from beginning of band	Yes
0x06	Seek down from end of band	Yes
0x07	Seek up from current frequency	Yes
0x08	Seek down from current frequency	Yes
0x09	Seek up from beginning of band for an HD signal	No
0x0A	Seek down from end of band for an HD signal	No
0x0B	Seek up from current frequency for an HD signal	No
0x0C	Seek down from current frequency for an HD signal	No
0x0D	Seek up from beginning of band for an HD signal	Yes

ID	Operation	Use RSSI threshold
0x0E	Seek down from end of band for an HD signal	Yes
0x0F	Seek up from current frequency for an HD signal	Yes
0x10	Seek down from current frequency for an HD signal	Yes
0x11–0xFF	Reserved	N/A

An analog seek operation (operation 0x01–0x08) will seek any frequency that contains a valid signal including HD signals. An HD seek operation (operation 0x09–0x10) skips channels that are not HD and stops only on channels with HD signals present. The HD seek completes when either of two conditions is satisfied:

- An HD channel that satisfies the criteria of the tuner’s seek function was located within the band. This may result in moving one or more channel spacings and wrapping around one end of the band.
- No HD channel that satisfies the criteria of the tuner’s seek function was located within the band, and the seek has traversed the entire band and wrapped back to the original tuner frequency.

A seek operation using an RSSI threshold completes when either of two conditions is satisfied:

- A channel was located within the band that satisfies the minimum RSSI threshold level.
- No channel was located within the band that satisfies the minimum the RSSI threshold level. The seek has traversed the entire band and wrapped back to the beginning tuner frequency without locating a valid channel. If no channel is found, it may indicate that the threshold is too high for the current radio reception area.

A seek operation using no RSSI threshold completes when either of two conditions is satisfied:

- A channel was located within the band that satisfies the criteria of the tuner’s seek function. This may result in moving one or more channel spacings and wrapping around at the band ends.
- No channel was located within the band that satisfies the criteria of the tuner’s seek function and the seek has traversed the entire band and wrapped back to the beginning tuner frequency without locating a valid channel.

An **ACK** command with command pending status is returned for seek operations requiring more than 100 ms to complete. It indicates the maximum time required for the device to complete the requested scan type. When the requested seek operation is completed (either successfully or unsuccessfully), the device must respond with a **TunerSeekDone** command indicating the seek operation status. If the device does not support tuner seek (with RSSI) operations or if tuner power is off, an **ACK** command with error status is returned.

When a cancel seek operation is requested and a seek operation is active, the seek operation stops at the current seek channel and the device responds with a **TunerSeekDone** command indicating the frequency and RSSI of the channel.

## Command 0x13: TunerSeekDone

Direction: Device to iPod

In response to a `TunerSeekStart` command from an iPod, an RF tuner device must send this command to the iPod after the seek operation has finished. It reports the current tuner frequency, which is assumed to be the result of the seek operation. If the device supports a tuner channel RSSI indication capability (as reported by a previous `RetTunerCaps` command), the `rssiLevel` value shows the current channel's RSSI signal strength level. If no channel was found, a tuner frequency value of 0xFFFFFFFF must be reported. The received signal strength value must be normalized to a range from 0x00 (no signal or device does not support RSSI indication) to 0xFF (maximum signal strength).

If an HD signal seek was requested, the tuner should tune to the next frequency containing HD content but it should not select an analog or HD program service. The iPod will perform the program selection operation separately. See ["Command 0x28: RetHDProgramService"](#) (page 256) and ["Command 0x29: SetHDProgramService"](#) (page 257).

**Table 3-124** TunerSeekDone packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x07	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x13	Command ID: TunerSeekDone
5	0xNN	Tuner frequency in kilohertz (bits 31:24)
6	0xNN	Tuner frequency in kilohertz (bits 23:16)
7	0xNN	Tuner frequency in kilohertz (bits 15:8)
8	0xNN	Tuner frequency in kilohertz (bits 7:0)
9	0xNN	Tuner channel RSSI received signal strength value
10	0xNN	Checksum

## Command 0x14: GetTunerStatus

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to get its current tuner status state; in response, the device must send a `RetTunerStatus` command. The command can be used to poll the device's status if the device does not support status change notifications. After the device's status is reported its status bits must be cleared, so that an immediate reread will return no active status.

**Table 3-125** GetTunerStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x14	Command ID: <code>GetTunerStatus</code>
5	0xE3	Checksum

## Command 0x15: `RetTunerStatus`

Direction: Device to iPod

This command is sent by an RF tuner device in response to a `GetTunerStatus` command received from an iPod. The tuner status bits returned are valid only if the status capability bits returned by a previous `RetTunerCaps` command were set. If the tuner power is off, the last active tuner status information should be returned.

**Table 3-126** `RetTunerStatus` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x15	Command ID: <code>RetTunerStatus</code>
5	0xNN	Tuner status bits (see <a href="#">Table 3-127</a> (page 242))
6	0xNN	Checksum

**Note:** An accessory that declares support for HD radio by setting its capability bit 04 must return valid RF tuner status bits; see [Table 3-100](#) (page 227).

**Table 3-127** RF tuner status bits

Bit	Description
0	RDS/RBDS data is received, ready to read
1	Tuner channel RSSI level has changed
2	Stereo source indicator state; 1 for a stereo signal source, 0 for monophonic

Bit	Description
3	HD signal present
4	HD digital audio present
5	HD data is received; ready to read
7:6	Reserved

## Command 0x16: GetStatusNotifyMask

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to get the status notification mask from the device. This mask indicates which state changes will invoke a notification change command from the device. In response, the device must send the iPod a `RetStatusNotifyMask` command.

**Table 3-128** `GetStatusNotifyMask` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x16	Command ID: <code>GetStatusNotifyMask</code>
5	0xE1	Checksum

## Command 0x17: RetStatusNotifyMask

Direction: Device to iPod

This command must be returned by the device in response to the `GetStatusNotifyMask` command. The status notification mask indicates which state changes will invoke a notification change command from the device. However, its bit values are valid only if the corresponding capabilities bits returned by a previous `RetTunerCaps` command were set. A mask bit value of 1 indicates that notification is enabled; a value of 0 indicates notification is disabled or not supported.

**Table 3-129** `RetStatusNotifyMask` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x17	Command ID: RetStatusNotifyMask
5	0xNN	Status notification mask (see <a href="#">Table 3-130</a> (page 244))
6	0xNN	Checksum

**Note:** An accessory that declares support for HD radio by setting its capability bit 04 (see [Table 3-100](#) (page 227)) must support status notifications.

**Table 3-130** Status notification mask bits

Bit	Description
0	RDS/RBDS data-ready change notify enabled (ignored if SetRdsNotifyMask sets rdsMask to a value other than zero)
1	Tuner channel RSSI-level change notify enabled
2	Stereo indicator state change notify enabled
3	HD signal present notification enabled
4	HD digital audio present notification enabled
5	HD data ready notification enabled
7:6	Reserved

## Command 0x18: SetStatusNotifyMask

Direction: iPod to Device

The iPod sends this command to an RF tuner device to set the status change notification mask. The status notification mask indicates which state changes will invoke a notification change command from the device. However, its bit values are valid only if the corresponding capabilities bits returned by a previous RetTunerCaps command were set. A mask bit value of 1 indicates that notification is enabled; a value of 0 indicates notification is disabled or not supported.



**Note:** For RDS/RBDS data-ready changes, the `SetRdsNotifyMask` command can override the notification status mask bit. If the `SetRdsNotifyMask` notification mask is set with a nonzero mask, then any enabled RDS/RBDS data notifications will be sent using the `RdsReadyNotify` command instead of the `StatusChangeNotify` command. For HD data ready changes, the `SetHDDDataNotifyMask` command can override the HD data notification status mask bit. If the `SetHDDDataNotifyMask` notification mask is set with a nonzero mask, then any enabled HD data notifications must be sent using the `HDDDataReadyNotify` command instead of the `StatusChangeNotify` command.

**Table 3-131** SetStatusNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x18	Command ID: SetStatusNotifyMask
5	0xNN	Status notification mask (see <a href="#">Table 3-132</a> (page 245))
6	0xNN	Checksum

**Table 3-132** Status notification mask setting bits

Bit	Description
0	Enable RDS/RBDS data-ready change notification (ignored if <code>SetRdsNotifyMask</code> sets <code>rdsMask</code> to a value other than zero)
1	Enable tuner channel RSSI-level change notification
2	Enable stereo-indicator state change notification
3	Enable HD signal present notification
4	Enable HD digital audio present notification
5	Enable HD data ready notification
7:6	Reserved

## Command 0x19: StatusChangeNotify

Direction: Device to iPod

This command must be sent asynchronously by an RF tuner device to an attached iPod to report each enabled status change. The iPod enables specific RF tuner status change notifications using the `SetStatusNotifyMask` command. After a notification has been sent, the status bits should be automatically cleared so they are ready to receive the next status change.

**Note:** Tuner channel RSSI-level change and stereo-indicator state change notifications must not occur more than once every 100 ms.

**Table 3-133** StatusChangeNotify packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x19	Command ID: StatusChangeNotify
5	0xNN	Status change ID bits (see Table 3-134 (page 246))
6	0xNN	Checksum

**Note:** For RDS/RBDS data-ready changes, the `SetRdsNotifyMask` command can override the notification status mask bit. If the `SetRdsNotifyMask` notification mask is set with a nonzero mask, then any enabled RDS/RBDS data notifications will be sent using the `RdsReadyNotify` command instead of the `StatusChangeNotify` command. Similarly, for HD data-ready changes, the `SetHDDDataNotifyMask` command can override the HD data notification status mask bit. If the `SetHDDDataNotifyMask` notification mask is set with a nonzero mask, then any enabled HD data notifications will be sent using the `HDDDataReadyNotify` command instead of the `StatusChangeNotify` command.

**Table 3-134** Status change ID bits

Bit	Description
0	RDS/RBDS data-ready change (valid only if bit 0 of <code>statusMask</code> is 1 and <code>SetRdsNotifyMask</code> sets <code>rdsMask</code> to 0x0)
1	Tuner channel RSSI-level change
2	Stereo indicator state change
3	HD signal present
4	HD digital audio present
5	HD data is received; ready to read
7:6	Reserved

## Command 0x1A: GetRdsReadyStatus

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to get the device's current RDS/RBDS data-ready status. It can be used to poll the device's RDS/RBDS data-ready status without having to enable RDS/RBDS data-ready notifications. This command is usable only if the `RetTunerCaps` capability bits report has indicated that the device supports RDS/RBDS data reception and parsing. In response, the device must send a `RetRdsReadyStatus` command.

**Table 3-135** `GetRdsReadyStatus` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1A	Command ID: <code>GetRdsReadyStatus</code>
5	0xDD	Checksum

## Command 0x1B: RetRdsReadyStatus

Direction: Device to iPod

This command must be sent by an RF tuner device in response to a `GetRdsReadyStatus` command received from an iPod. Its status value indicates which RDS/RBDS data values are available to be read. All status bits must remain set on the device until the iPod sends a `GetRdsData` command to read the data. If the device does not support RDS/RBDS data, it must send an `rdsReady` data-ready status of 0 to indicate that no data is ready.

**Table 3-136** `RetRdsReadyStatus` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1B	Command ID: <code>RetRdsReadyStatus</code>
5	0xNN	RDS/RBDS data-ready status (bits 31:24) (see <a href="#">Table 3-137</a> (page 248) and <a href="#">Table 3-138</a> (page 248))

Byte number	Value	Comment
6	0xNN	RDS/RBDS data-ready status (bits 23:16)
7	0xNN	RDS/RBDS data-ready status (bits 15:8)
8	0xNN	RDS/RBDS data-ready status (bits 7:0)
9	0xNN	Checksum

**Table 3-137** RDS/RBDS data-ready status bits, parsed mode

Bit	Description
03-00	Reserved; set to zeros
04	RadioText (RT) data-ready
29-05	Reserved; set to zeros
30	Program Service Name (PSN) data-ready
31	Reserved; set to zero

**Table 3-138** RDS/RBDS data-ready status bits, raw mode

Bit	Description
04-00	Reserved
05	RDS/RBDS group data ready
31-06	Reserved

## Command 0x1C: GetRdsData

Direction: iPod to Device

This command is sent by an iPod to get raw or unparsed RDS/RBDS data from an RF tuner device that supports the RDS/RBDS data capability. If the device supports RDS/RBDS and has data-ready, it must send a `RetRdsData` command. If the device does not support RDS/RBDS or does not have the specified data type ready, it must return an `ACK` command with command failure status.

**Table 3-139** `GetRdsData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1C	Command ID: GetRdsData
5	0xNN	RDS/RBDS data type ( <code>rdsDataType</code> ) ID (see <a href="#">Table 3-140</a> (page 249) and <a href="#">Table 3-141</a> (page 249))
6	0xNN	Checksum

**Table 3-140** RDS/RBDS data type IDs, parsed mode

ID Bytes	Description
0x00–0x03	Reserved
0x04	RadioText (RT)
0x05–0x1D	Reserved
0x1E	Program Service Name (PSN)
0x1F–0xFF	Reserved

**Table 3-141** RDS/RBDS data type IDs, raw mode

ID Bytes	Description
0x00–0x04	Reserved
0x05	RDS/RBDS group data ready
0x06–0xFF	Reserved

## Command 0x1D: RetRdsData

---

Direction: Device to iPod

This command is sent by an RF tuner device that has an RDS/RBDS data group ready in response to a `GetRdsData` command received from an iPod. If the device does not support the RDS/RBDS capability or does not have the specified data type ready, it must return an `ACK` command with command failure status (command status 0x02).

**Table 3-142** `RetRdsData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1D	Command ID: RetRdsData
5	0xNN	rdsDataType (see <a href="#">Table 3-143</a> (page 250))
6	0xNN	rdsData (see <a href="#">Table 3-143</a> (page 250))
NN	0xNN	Checksum

**Table 3-143** rdsDataType bytes and rdsData formats

rdsDataType bytes	Description	rdsData format
0x00–0x03	Reserved	N/A
0x04	RadioText (RT)	charSet:1 (see <a href="#">Table 3-144</a> (page 250)), radioText:64
0x05–0x1D	Reserved	N/A
0x1E	Program Service Name (PSN)	charSet:1 (see <a href="#">Table 3-144</a> (page 250)), progSvcName:8
0x1F–0xFF	Reserved	N/A

**Note:** The maximum length of a Radio Text is 64 bytes and that of a Program Service Name is 8 bytes.

**Table 3-144** rdsData character set IDs

ID	Character set
0x00	Latin-based languages
0x01	Cyrillic- and Greek-based languages
0x02	Arabic- and Hebrew-based languages
0x03–0xFF	Reserved

When RDS/RBDS Raw mode is enabled, the data shown in [Table 3-145](#) (page 251) is returned for the rdsDataType x05 (RDS/RBDS group data ready).

**Table 3-145** RDS/RBDS group data-ready data

Byte	Description
0x00-0x01	16 bit value for Block A of RDS/RBDS group data
0x02-0x03	16 bit value for Block B of RDS/RBDS group data
0x04-0x05	16 bit value for Block C of RDS/RBDS group data
0x06-0x07	16 bit value for Block D of RDS/RBDS group data
0x08	Block errors byte (see <a href="#">Table 3-146</a> (page 251))

**Table 3-146** Block errors byte encoding

Byte	Description	
1:0	Block A error bits	See <a href="#">Table 3-147</a> (page 251)
3:2	Block B error bits	
5:4	Block C error bits	
7:6	Block D error bits	

**Table 3-147** Block error bit values

Bits	Description
00	No errors
01	1 to 2 errors; slight chance of uncorrected errors
10	3 to 5 errors; may have uncorrected errors
11	6 or more errors; uncorrectable block

## Command 0x1E: GetRdsNotifyMask

---

Direction: iPod to Device

This command is sent by an iPod to get an RF tuner device's current RDS/RBDS data notification mask (`rdsMask`). In response, the device must send a `RetRdsNotifyMask` command with the RDS/RBDS data notification mask. This command is valid only if the RDS/RBDS data-ready capability bit was set in a previous `RetTunerCaps` command.

**Table 3-148** GetRdsNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1E	Command ID: GetRdsNotifyMask
5	0xD9	Checksum

## Command 0x1F: RetRdsNotifyMask

---

Direction: Device to iPod

This command is sent by an RF tuner device in response to a GetRdsNotifyMask command sent by an iPod. The RDS/RBDS mask (`rdsMask`) indicates which asynchronous data notifications should be sent by the device when data is ready. For each data type bit, 1 means that data-ready notification is enabled and 0 means that notification is disabled. For enabled RDS/RBDS data types, the device must send `RdsReadyNotify` commands to the iPod when the associated data becomes available.

**Table 3-149** RetRdsNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x1F	Command ID: RetRdsNotifyMask
5	0xNN	RDS/RBDS data change notification mask (bits 31:24) (see <a href="#">Table 3-150</a> (page 253) and <a href="#">Table 3-151</a> (page 253))
6	0xNN	RDS/RBDS data change notification mask (bits 23:16)
7	0xNN	RDS/RBDS data change notification mask (bits 15:8)
8	0xNN	RDS/RBDS data change notification mask (bits 7:0)
9	0xNN	Checksum



**Table 3-150** RDS/RBDS data change notification mask bits, parsed mode

Bit	Description
03-00	Reserved
04	RadioText (RT)
29-05	Reserved (must be set to zeros)
30	Program Service Name (PSN)
31	Reserved (must be set to 0)

**Table 3-151** RDS/RBDS data change notification mask bits, raw mode

Bit	Description
04-00	Reserved
05	RDS/RBDS group data
31-06	Reserved

## Command 0x20: SetRdsNotifyMask

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to set the device's RDS/RBDS data-ready notification mask (`rdsMask`). When notification bits are enabled, the device must send an `RdsReadyNotify` command with the associated RDS/RBDS data. For each RDS/RBDS data type bit, 1 enables data-ready notification and 0 disables notification.

**Note:** For RDS/RBDS data-ready changes, the `SetRdsNotifyMask` command can override the notification status mask bit. If the `SetRdsNotifyMask` notification mask is set with a nonzero mask, then any enabled RDS/RBDS data notifications will be sent using the `RdsReadyNotify` command instead of the `StatusChangeNotify` command.

**Table 3-152** `SetRdsNotifyMask` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x20	Command ID: <code>SetRdsNotifyMask</code>

Byte number	Value	Comment
5	0xNN	RDS/RBDS data change notification mask (bits 31:24) (see <a href="#">Table 3-153</a> (page 254) and <a href="#">Table 3-154</a> (page 254))
6	0xNN	RDS/RBDS data change notification mask (bits 23:16)
7	0xNN	RDS/RBDS data change notification mask (bits 15:8)
8	0xNN	RDS/RBDS data change notification mask (bits 7:0)
9	0xNN	Checksum

**Table 3-153** RDS/RBDS data change notification mask setting bits, parsed mode

Bit	Description
03-00	Reserved
04	RadioText (RT)
29-05	Reserved
30	Program Service Name (PSN)
31	Reserved

**Table 3-154** RDS/RBDS data change notification mask setting bits, raw mode

Bit	Description
04-00	Reserved
05	RDS/RBDS group data
31-06	Reserved

## Command 0x21: RdsReadyNotify

---

Direction: Device to iPod

This command must be sent by an RF tuner device when RDS/RBDS data is ready, the associated `SetRdsNotifyMask` bit is set, and the device's capability bit from a previous `RetTunerCaps` command indicates that the device supports RDS/RBDS status notifications. After a notification command is sent, the device's associated RDS/RBDS data-ready status bit must be cleared.

**Table 3-155** `RdsReadyNotify` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x21	Command ID: <code>RdsReadyNotify</code>
5	0xNN	<code>rdsDataType</code> (for RDS data types, see <a href="#">Table 3-143</a> (page 250); for HD data types, see <a href="#">Table 3-167</a> (page 261))
6	0xNN	<code>rdsData</code> (for RDS data types, see <a href="#">Table 3-143</a> (page 250); for HD data types, see <a href="#">Table 3-167</a> (page 261))
NN	0xNN	Checksum

## Command 0x25: GetHDProgramServiceCount

Direction: iPod to Device

This command is sent by an iPod to an RF Tuner device to get the count of HD program services broadcast at the current tuner frequency. In response, the device must send a `RetHDProgramServiceCount` command with the count of HD program services available. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

**Table 3-156** `GetHDProgramServiceCount` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x25	Command ID: <code>GetHDProgramServiceCount</code>
5	0xD2	Checksum

## Command 0x26: RetHDProgramServiceCount

Direction: Device to iPod

This command is sent by an RF Tuner device in response to a `GetHDProgramServiceCount` command sent by an iPod. The command returns the count of HD program services broadcast at the current tuner frequency. The command returns a count of 0 if there are no HD program services available. HD programs services must

be indexed from 1 to the count value, 1 being the main program service. The Analog Program Exists field indicates whether the station has analog programming or not. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

**Table 3-157** `RetHDProgramServiceCount` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x26	Command ID: <code>RetHDProgramServiceCount</code>
5	0xNN	HD program service count (0-8)
6	0xNN	Analog Program Exists (0 = no, 1 = yes)
7	0xNN	Checksum

## Command 0x27: `GetHDProgramService`

---

Direction: iPod to Device

This command is sent by an iPod to an RF Tuner device to get the current tuned HD program service. In response, the device must send a `RetHDProgramService` command with the current HD program service. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

**Table 3-158** `GetHDProgramService` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x27	Command ID: <code>GetHDProgramService</code>
5	0xD0	Checksum

## Command 0x28: `RetHDProgramService`

---

Direction: Device to iPod

This command is sent by an RF Tuner device in response to a `GetHDProgramService` command sent by an iPod. The value returned is the tuned HD program service, 0 if the analog program is currently tuned, or 0xFF if audio decoding and output is currently disabled. The output may be disabled if a `SetHDProgramService` command was sent with parameter 0xFF or if the radio has just been tuned to a station's frequency. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

**Table 3-159** `RetHDProgramService` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x28	Command ID: <code>RetHDProgramService</code>
5	0xNN	HD program service index (0x00-0x08 or 0xFF)
6	0xNN	Checksum

## Command 0x29: `SetHDProgramService`

Direction: iPod to Device

This command is sent by an iPod to an RF Tuner device to tune to an HD program service. Tuning to program service 0 disables HD tuning and switches back to analog tuning, if it is available. Tuning to program service 0xFF disables all audio decoding and output. This command lets the iPod retrieve information for all available HD programs before selecting a program's audio to be decoded and presented. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

Selecting an analog program on an HD digital-only station or attempting to select a nonexistent HD program constitutes an invalid program selection. In this case, the accessory should send the iPod an `ACK` command with Bad Parameter status.

**Table 3-160** `SetHDProgramService` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x29	Command ID: <code>SetHDProgramService</code>
5	0xNN	HD program service index (0x00-0x08 or 0xFF)

Byte number	Value	Comment
6	0xNN	Checksum

## Command 0x2A: GetHDDataReadyStatus

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to get the device's current HD data-ready status. It can be used to poll the device's HD data-ready status without having to enable HD data-ready notifications. In response, the device must send a `RetHDDataReadyStatus` command. This command is only valid if the HD capable bit was set in a previous `RetTunerCaps` command.

**Note:** Only parsed mode is supported for HD Data.

**Table 3-161** GetHDDataReadyStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2A	Command ID: GetHDDataReadyStatus
5	0xCD	Checksum

## Command 0x2B: RetHDDataReadyStatus

Direction: Device to iPod

This command must be sent by an RF tuner device in response to a `GetHDDataReadyStatus` command received from an iPod. Its status value indicates which HD data values are available to be read. All status bits must remain set on the device until the iPod sends a `GetHDData` command to read the data.

**Table 3-162** RetHDDataReadyStatus packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet

Byte number	Value	Comment
3	0x07	Lingo ID: RF Tuner lingo
4	0x2B	Command ID: RetHDDataReadyStatus
5	0xNN	HD data-ready status (bits 31:24) (see <a href="#">Table 3-163</a> (page 259))
6	0xNN	HD data-ready status (bits 23:16)
7	0xNN	HD data-ready status (bits 15:8)
8	0xNN	HD data-ready status (bits 7:0)
9	0xNN	Checksum

**Table 3-163** HD data-ready status bits

Bit	Description
0	PSD data ready
1	Reserved
2	SIS Station ID number data-ready
3	SIS Station Name (short) data-ready
4	SIS Station Name (long) data-ready
5	SIS ALFN data-ready
6	SIS Station Location data-ready
7	SIS Station Message data-ready
8	SIS Slogan data-ready
9	SIS Parameter Message data-ready
31:10	Reserved

## Command 0x2C: GetHDData

---

Direction: iPod to Device

This command is sent by an iPod to get HD data from an RF tuner device. If the device supports HD and has data-ready, it must send a RetHDData command. If the device does not support HD or does not have the specified data type ready, it must return an ACK command with command failure status. This command is valid only if the HD capable bit was set in a previous RetTunerCaps command.

**Note:** Only parsed mode is supported for HD Data.

**Table 3-164** GetHDData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2C	Command ID: GetHDData
5	0xNN	HD data type; see <a href="#">Table 3-165</a> (page 260)
6	0xNN	Checksum

**Table 3-165** HD data type IDs

Byte number	Description
0x00	PSD data
0x01	Reserved
0x02	SIS Station ID number data
0x03	SIS Station Name (short) data
0x04	SIS Station Name (long) data
0x05	SIS ALFN data
0x06	SIS Station Location data
0x07	SIS Station Message data
0x08	SIS Slogan
0x09	SIS Parameter Message data
0x0A-0xFF	Reserved

## Command 0x2D: RetHDData

Direction: Device to iPod



This command is sent by an RF Tuner device that has HD data ready in response to a `GetHDDData` command. If the device does not have the specified data type ready, it must return an `ACK` command with command failure status (command status 0x02). On sending this command, the accessory device must keep clear its internal HD data-ready status for the data type being read until the next time data is ready.

**Table 3-166** RetHDDData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2D	Command ID: RetHDDData
5	0xNN	HDDDataType (see <a href="#">Table 3-167</a> (page 261))
6	0xNN	HDDData
NN	0xNN	Checksum

**Table 3-167** HDDDataType type IDs and formats

HDDDataType ID	HDDData format
0x00	8 Bit integer index, 8 Bit integer max index, 8 bit HD program index, ID3 Frame data (see <a href="#">Table 3-168</a> (page 262))
0x01	N/A
0x02	32-bit integer station ID
0x03	8-bit character encoding type, up to 12 bytes (4 or 7 in USA) of short station name (see <a href="#">Table 3-169</a> (page 262))
0x04	0-56 characters station name (UTF-7)
0x05	32-bit ALFN
0x06	27-bit GPS latitude/longitude encoded in lower bits of 32 bit data
0x07	8-bit character encoding type, 0-190 bytes station message (see <a href="#">Table 3-169</a> (page 262))
0x08	8-bit character encoding type, 2-96 bytes station slogan (see <a href="#">Table 3-169</a> (page 262))
0x09	6-bit index encoded in lower bits of 8 bit data, 16-bit parameter
0x0A–0xFF	N/A

**Table 3-168** PSD data format

HDDData type ID	Description
0x00	Current ID3 Frame index (0 = first)
0x01	Last Index of ID3 Frame data
0x02	HD Program Index (1-8)
0x03–0xNN	ID3 Frame data, ID3 tag version 2.3.0. The first byte of the ID3 tag's text information contains the encoding type.

**Table 3-169** 8-bit character encoding type formats

Encoding type ID	Description
0x00	ISO/IEC 8859-1:1998
0x01–0x03	Reserved
0x04	ISO/IEC 10646-1:2000, UCS-2 (Little-endian)
0x05–0xFF	Reserved

## Command 0x2E: GetHDDDataNotifyMask

Direction: iPod to Device

This command is sent by an iPod to get an RF tuner device's current HD data notification mask. In response, the device must send a `RetHDDDataNotifyMask` command with the HD data notification mask. This command is valid only if the HD capability bit was set in a previous `RetTunerCaps` command.

**Table 3-170** GetHDDDataNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2E	Command ID: GetHDDDataNotifyMask
5	0xC9	Checksum

## Command 0x2F: RetHDDDataNotifyMask

Direction: Device to iPod

This command is sent by an RF tuner device in response to a `GetHDDDataNotifyMask` command sent by an iPod. The HD mask indicates which asynchronous data notifications should be sent by the device when data is ready. For each data type bit, 1 means that data-ready notification is enabled and 0 means that notification is disabled. For enabled HD data types, the device must send `HDDataReadyNotify` commands to the iPod when the associated data becomes available.

**Table 3-171** RetHDDDataNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x2F	Command ID: RetHDDDataNotifyMask
5	0xNN	HD data change notification mask (bits 31:24) (see <a href="#">Table 3-172</a> (page 263))
6	0xNN	HD data change notification mask (bits 23:16)
7	0xNN	HD data change notification mask (bits 15:8)
8	0xNN	HD data change notification mask (bits 7:0)
9	0xNN	Checksum

**Table 3-172** HD data change notification mask bits

Bit	Description
0	PSD data ready
1	Reserved
2	SIS Station ID number data-ready
3	SIS Station Name (short) data-ready
4	SIS Station Name (long) data-ready
5	SIS ALFN data-ready
6	SIS Station Location data-ready
7	SIS Station Message data-ready

Bit	Description
8	SIS Slogan data-ready
9	SIS Parameter Message data-ready
31:10	Reserved

## Command 0x30: SetHDDDataNotifyMask

Direction: iPod to Device

This command is sent by an iPod to an RF tuner device to set the device's HD data-ready notification mask. When notification bits are enabled, the device must send an `HDDDataReadyNotify` command with the associated HD data. For each HD data type bit, 1 enables data-ready notification and 0 disables notification. This command is valid only if the HD capability bit was set in a previous `RetTunerCaps` command.

**Table 3-173** SetHDDDataNotifyMask packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x30	Command ID: SetHDDDataNotifyMask
5	0xNN	HD data change notification mask (bits 31:24) (see <a href="#">Table 3-174</a> (page 264))
6	0xNN	HD data change notification mask (bits 23:16)
7	0xNN	HD data change notification mask (bits 15:8)
8	0xNN	HD data change notification mask (bits 7:0)
9	0xNN	Checksum

**Table 3-174** HD data change notification mask setting bits

Bit	Description
0	PSD data ready
1	Reserved
2	SIS Station ID number data-ready
3	SIS Station Name (short) data-ready

Bit	Description
4	SIS Station Name (long) data-ready
5	SIS ALFN data-ready
6	SIS Station Location data-ready
7	SIS Station Message data-ready
8	SIS Slogan data-ready
9	SIS Parameter Message data-ready
31:10	Reserved

## Command 0x31: HDDataReadyNotify

Direction: Device to iPod

This command must be sent by an RF tuner device when HD data is ready, the associated `SetHDDataNotifyMask` bit is set, and the device's capability bit from a previous `RetTunerCaps` command indicates that the device supports HD status notifications. After a notification command is sent, the device's associated HD data-ready status bit must be cleared.

**Table 3-175** HDDataReadyNotify packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x07	Lingo ID: RF Tuner lingo
4	0x31	Command ID: HDDataReadyNotify
5	0xNN	HDDataType (see <a href="#">Table 3-176</a> (page 265))
6–0xNN	0xNN	HD data (see <a href="#">Table 3-177</a> (page 266))
NN	0xNN	Checksum

**Table 3-176** HD data types

Byte	Description
0x00	PSD data
0x01	Reserved

Byte	Description
0x02	SIS Station ID number data
0x03	SIS Station Name (short) data
0x04	SIS Station Name (long) data
0x05	SIS ALFN data
0x06	SIS Station Location data
0x07	SIS Station Message data
0x08	SIS Slogan data
0x09	SIS Parameter Message data
0x0A–0xFF	Reserved

**Table 3-177** HD Data Type type IDs and formats

HD data type ID	HDData format
0x00	8 Bit integer index, 8 Bit integer max index, 8 bit HD program index, ID3 Frame data (see <a href="#">Table 3-178</a> (page 266))
0x01	N/A
0x02	32-bit integer station ID
0x03	8-bit character encoding type, up to 12 bytes (4 or 7 in USA) of short station name (see <a href="#">Table 3-179</a> (page 267))
0x04	0-56 characters station name (UTF-7)
0x05	32-bit ALFN
0x06	27-bit GPS latitude/longitude encoded in lower bits of 32 bit data
0x07	8-bit character encoding type, 0-190 bytes station message (see <a href="#">Table 3-179</a> (page 267))
0x08	8-bit character encoding type, 2-96 bytes station slogan (see <a href="#">Table 3-179</a> (page 267))
0x09	6-bit index encoded in lower bits of 8 bit data, 16-bit parameter
0x0A–0xFF	N/A

**Table 3-178** PSD data format

HDDataType ID	Description
0x00	Current ID3 Frame index (0 = first)

HDDataType ID	Description
0x01	Last Index of ID3 Frame data
0x02	HD Program Index (1-8)
0x03–0x <i>NN</i>	ID3 Frame data, ID3 tag version 2.3.0. The first byte of the ID3 tag's text information contains the encoding type.

**Table 3-179** 8-bit character encoding type formats

Encoding type ID	Description
0x00	ISO/IEC 8859-1:1998
0x01–0x03	Reserved
0x04	ISO/IEC 10646-1:2000, UCS-2 (Little-endian)
0x05–0xFF	Reserved

## Sample HD Command Sequences

This section presents three typical examples of using RF Tuner lingo commands with HD radio.

### Initializing HD Radio

This example sets up HD Radio using RF Tuner lingo commands.

**Table 3-180** Example of HD radio setup

Command	Direction	Data	Comments
GetTunerCaps	iPod to Dev	No Data	Request RF Tuner accessory's supported capabilities.
RetTunerCaps	Dev to iPod	0x07FC0712	Return 32-bit field indicating the accessory's capabilities (FM band US, HD Radio, Tuner power control, status change notification, FM resolution 200 kHz, Tuner seek capable, Tuner seek RSSI threshold capable, Force monophonic mode capable, Stereo blend capable, FM Tuner deemphasis select capable, AM resolution 10 kHz, RDS/RBDS data capable, Tuner channel RSSI indication capable, Stereo source indicator capable).
SetTunerCtrl	iPod to Dev	0x01	Tuner power on, status change notification off, raw mode off.
ACK	Dev to iPod	0x00	Acknowledge SetTunerCtrl

Command	Direction	Data	Comments
SetStatusNotifyMask	iPod to Dev	0x38	Set status notification for HD signal present, HD digital audio present, HD data ready.
ACK	Dev to iPod	0x00	Acknowledge SetStatusNotifyMask
SetHDDDataNotifyMask	iPod to Dev	0x00000009	Set HD data ready notification for Station Short Name and PSD Data.
ACK	Dev to iPod	0x00	Acknowledge SetHDDDataNotifyMask
SetTunerMode	iPod to Dev	0x00	Set FM Tuner resolution 200kHz, stereo allowed, no stereo blend, FM Tuner de-emphasis 75 µsec, AM Tuner resolution 10kHz.
ACK	Dev to iPod	0x00	Acknowledge SetTunerMode

## HD Radio Tuning and Reception

Using RF Tuner lingo commands, this example tunes to an HD radio station, collects data on it, and responds to a loss of digital audio.

**Table 3-181** Example of HD radio tuning and reception

Command	Direction	Data	Comments
SetTunerBand	iPod to Dev	0x01	Set Tuner band to FM US.
ACK	Dev to iPod	0x00	Acknowledge SetTunerBand
SetTunerFreq	iPod to Dev	97700	Set FM Tuner frequency to 97.7 MHz
RetTunerFreq	Dev to iPod	97700, 31	Return FM Tuner frequency set to 97.7 MHz with RSSI level of 31.
SetTunerSeekRssi	iPod to Dev	8	Set the tuner seek RSSI threshold to 8.
ACK	Dev to iPod	0x00	Acknowledge SetTunerSeekRssi
TunerSeekStart	iPod to Dev	0x0F	Start tuner seek up from current frequency while checking for RSSI threshold.
ACK	Dev to iPod	0x00	Acknowledge TunerSeekStart
TunerSeekDone	Dev to iPod	106900, 30	Tuner seek has finished with tuned frequency of 106.9 MHz and RSSI level of 30.
GetHDProgramServiceCount	iPod to Dev	No data	Retrieve number of HD Program Services available.



Command	Direction	Data	Comments
RetHDProgramService-Count	Dev to iPod	3,1	3 HD Program services and the analog programming are available.
SetHDProgramService	iPod to Dev	1	Start decode and playback of HD program service #1.
ACK	Dev to iPod	0x00	Acknowledge SetHDProgramService
Collect info on HD programs, as shown in "HD Radio Service Management" (page 270). Info collection could have been performed before tuning by using SetHDDataNotifyMask; see "Initializing HD Radio" (page 267).			
Set notification:			
SetStatusNotifyMask	iPod to Dev	0x18	Set status notification for HD signal present, HD digital audio present.
ACK	Dev to iPod	0x00	Acknowledge SetStatusNotifyMask
Retrieve HD signal and audio status:			
StatusChangeNotify	Dev to iPod	0x18	Status change notification: HD signal and HD audio present.
GetHDProgramService-Count	iPod to Dev	No data	Retrieve number of HD Program Services available.
RetHDProgramService-Count	Dev to iPod	2,1	2 HD Program services and the analog programming are available.
SetHDProgramService	iPod to Dev	1	Start decode/playback of HD program service #1.
ACK	Dev to iPod	0x00	Acknowledge SetHDProgramService
Respond to loss of HD digital audio:			
StatusChangeNotify	Dev to iPod	0x08	Status change notification: HD Signal present, HD digital audio and data not present.
GetHDProgramService-Count	iPod to Dev	No data	Retrieve number of HD Program Services available.
RetHDProgramService-Count	Dev to iPod	0,1	Zero HD Program services and the analog programming are available.
SetHDProgramService	iPod to Dev	0	Start playback of analog programming.
ACK	Dev to iPod	0x00	Acknowledge SetHDProgramService

## HD Radio Service Management

This example retrieves PSD and other data for HD radio, using RF Tuner lingo commands.

**Table 3-182** Example of getting PSD and name data

Command	Direction	Data	Comments
SetHDDDataNotifyMask	iPod to Dev	0x00000009	Set HD data ready notification for Station Short Name and PSD Data.
ACK	Dev to iPod	0x00	Acknowledge SetHDDDataNotifyMask
HDDDataReadyNotify	Dev to iPod	0x03, 1, NNNN	Station Short Name received, ISO 8859-1.
HDDDataReadyNotify	Dev to iPod	0x00, 0, 1, 1, NNNN	PSD ID3 Frame data 1 of 2 received for program 1.
HDDDataReadyNotify	Dev to iPod	0x00, 1, 1, 1, NNNN	PSD ID3 Frame data 2 of 2 received for program 1.
HDDDataReadyNotify	Dev to iPod	0x00, 0, 0, 2, NNNN	PSD ID3 Frame data 1 of 1 received for program 2.
GetHDDDataReadyStatus	iPod to Dev	No Data	Ask accessory to return HD data ready status.
RetHDDDataReadyStatus	Dev to iPod	0x00000009	Station Short Name and PSD data available.
GetHDDData	iPod to Dev	0x03	Ask accessory for Station Short Name.
RetHDDData	Dev to iPod	0x03, 1, NNNN	Station Short Name, ISO 8859-1.
GetHDDData	iPod to Dev	0x00	Ask accessory for PSD Data.
RetHDDData	Dev to iPod	0x00, 0, 1, 1, NNNN	PSD ID3 Frame data 1 of 2 received for program 1.
RetHDDData	Dev to iPod	0x00, 1, 1, 1, NNNN	PSD ID3 Frame data 2 of 2 received for program 1.
GetHDDDataReadyStatus	iPod to Dev	No Data	Ask accessory to return HD data ready status.
RetHDDDataReadyStatus	Dev to iPod	0x00000001	PSD data available.
GetHDDData	iPod to Dev	0x00	Ask accessory for PSD Data.
RetHDDData	Dev to iPod	0x00, 0, 0, 2, NNNN	PSD ID3 Frame data 1 of 1 received for program 2.
GetHDDDataReadyStatus	iPod to Dev	No Data	Ask accessory to return HD data ready status.
RetHDDDataReadyStatus	Dev to iPod	0x00000000	No more data available.

## Lingo 0x08: Accessory Equalizer Lingo

iPods may be connected to accessory devices such as boom boxes or amplifiers that are capable of frequency response equalization. The Accessory Equalizer lingo, number 0x08, lets the iPod control the Equalizer Settings of the accessory device.

**Note:** The Accessory Equalizer lingo requires device authentication for all communication transports (serial or USB).

The Accessory Equalizer lingo is similar to the Display Remote Equalizer lingo, but the command direction is reversed: the iPod is the master, initiating commands to which the device responds. The iPod can query the total count of device Equalizer Settings and the UTF-8 name for each one; it can then get or set each Equalizer Setting on the accessory device.

To support this lingo, the iPod application software will add an iPod menu item for accessory equalizer selection and control.

### Equalizer Setting Requirements

To be compatible with the Accessory Equalizer lingo, an accessory device must observe these rules:

- The accessory must have an Equalizer Setting with an index of 0x00 and this must be the accessory equalizer's off/none setting.
- The Equalizer Index and Equalizer Setting count fields are 1 byte in size, limiting the setting count to a maximum of 255, including the required index 0x00 setting.
- Devices supporting this lingo must support at least two Equalizer Settings: off/none and at least one other setting. If fewer than two settings are returned by the device, the iPod will not present equalizer menu options to the user.

### Accessory Equalizer Lingo Commands

Table 3-183 (page 271) lists the commands included in the Accessory Equalizer lingo.

**Table 3-183** Accessory Equalizer lingo command summary

Command	ID	Direction	Data length	Protocol version	Authentication required
ACK	0x00	Dev to iPod	4	1.00	Yes
GetCurrentEQIndex	0x01	iPod to Dev	2	1.00	Yes
RetCurrentEQIndex	0x02	Dev to iPod	3	1.00	Yes
SetCurrentEQIndex	0x03	iPod to Dev	3	1.00	Yes
GetEQSettingCount	0x04	iPod to Dev	2	1.00	Yes

Command	ID	Direction	Data length	Protocol version	Authentication required
RetEQSettingCount	0x05	Dev to iPod	3	1.00	Yes
GetEQIndexName	0x06	iPod to Dev	3	1.00	Yes
RetEQIndexName	0x07	Dev to iPod	0xNN	1.00	Yes
Reserved	0x08–0xFF	N/A	N/A	N/A	N/A

## Command History of the Accessory Equalizer Lingo

[Table 3-184](#) (page 272) shows the history of command changes in the accessory equalizer lingo:

**Table 3-184** Accessory Equalizer lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x07	Accessory device Equalizer Index, count, and name support

## Command 0x00: ACK

Direction: Device to iPod

This command is sent by the device in response to a command sent from the iPod. An ACK response is sent when a bad parameter is received, an unsupported/invalid command is received, or a command completed but did not return any data. See [Table 3-185](#) (page 272).

**Table 3-185** ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x00	Command ID: ACK
5	0xNN	Status of command received (see <a href="#">Table 3-28</a> (page 169))
6	0xNN	ID of the command for which the response is being sent
7	0xNN	Checksum

## Command 0x01: GetCurrentEQIndex

Direction: iPod to Device

This command is sent by the iPod to request the current Equalizer Setting from the device. In response, the device sends a `RetCurrentEQIndex` command with the current Equalizer Index.

The timeout for this command is 2500 ms (2.5 seconds). If the device does not respond within the allotted time, the iPod will retry up to three times. If the device does not respond within the specified time and retry count, the command will fail.

**Table 3-186** `GetCurrentEQIndex` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x01	Command ID: <code>GetCurrentEQIndex</code>
5	0xF5	Checksum

## Command 0x02: RetCurrentEQIndex

Direction: Device to iPod

This command is sent by the device in response to the `GetCurrentEQIndex` command received from the iPod. The Equalizer Index returned may range from 0 (reserved for the off/none Equalizer Setting) to 254 (the maximum possible Equalizer Index). Devices are not required to support 255 settings; they may support as few as the two minimum required settings.

**Table 3-187** `RetCurrentEQIndex` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x02	Command ID: <code>RetCurrentEQIndex</code>
5	0xNN	eqIndex: The current accessory Equalizer Index. See <a href="#">Table 3-188</a> (page 274).
6	0xNN	Checksum

**Table 3-188** Device Equalizer Setting indices

Index	Description
0x00	Equalizer off/none
0x01	First Equalizer Setting
0xNN	Last Equalizer Setting (accessory-dependent maximum index)

## Command 0x03: SetCurrentEQIndex

Direction: iPod to Device

This command is sent by the iPod to select an accessory Equalizer Index from the supported range. The valid range is from 0 to one less than the Equalizer Setting count returned by the `RetEQSettingCount` command. See [Table 3-188](#) (page 274).

**Note:** Accessories may receive duplicate `SetCurrentEQIndex` commands at any time and must handle them correctly.

The timeout for this command is 3000 ms (3.0 seconds). If the device does not respond within the allotted time, the iPod will retry up to three times. If the device does not respond within the specified time and retry count, the command will fail.

**Table 3-189** SetCurrentEQIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x03	Command ID: SetCurrentEQIndex
5	0xNN	eqIndex: the selected accessory Equalizer Index
6	0xNN	Checksum

## Command 0x04: GetEQSettingCount

Direction: iPod to Device

This command is sent by the iPod to determine how many Equalizer Settings the device supports. In response, the device sends a `RetEQSettingCount` command with the count of Equalizer Settings supported.

The timeout for this command is 3000 ms (3.0 seconds). If the device does not respond within the allotted time, the iPod will retry up to three times. If the device does not respond within the specified time and retry count, the command will fail.

**Table 3-190** GetEQSettingCount packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x04	Command ID: GetEQSettingCount
5	0xF2	Checksum

## Command 0x05: RetEQSettingCount

Direction: Device to iPod

This command is sent by the device in response to a GetEQSettingCount command from the iPod. To support the Accessory Equalizer lingo, a device must report a minimum count of 0x02 (equalizer off/none plus one other setting) and may support a maximum count of 0xFF (equalizer off/none plus 254 other settings).

**Table 3-191** RetEQSettingCount packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x05	Command ID: RetEQSettingCount
5	0xNN	eqCount: the count of accessory equalizer indices. See <a href="#">Table 3-192</a> (page 275).
6	0xNN	Checksum

**Table 3-192** RetEQSettingCount parameter values

Range	Comment
0x00–0x01	Invalid equalizer count (minimum count is 0x02)

Range	Comment
0x02–0xFF	Valid equalizer count range

## Command 0x06: GetEQIndexName

Direction: iPod to Device

This command is sent by the iPod to obtain the name string associated with a specified Equalizer Index. In response, the device sends a `RetEQIndexName` command with the same Equalizer Index and the associated Equalizer Setting name string.

The timeout for this command is 3000 ms (3.0 seconds). If the device does not respond within the allotted time, the iPod will retry up to three times. If the device does not respond within the specified time and retry count, the command will fail.

**Table 3-193** `GetEQIndexName` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x06	Command ID: <code>GetEQIndexName</code>
5	0xNN	<code>eqIndex</code> : the selected accessory Equalizer Index
6	0xNN	Checksum

## Command 0x07: RetEQIndexName

Direction: Device to iPod

This command is sent by the device in response to a `GetEQIndexName` command received from the iPod. The `eqIndex` byte is the same index that was sent by the `GetEQIndexName` command. The `eqName` string is a null-terminated UTF-8 character array. The array length must not exceed 32 characters plus a null terminator character. This length limit minimizes truncation of the Equalizer Setting name when it is displayed in the iPod accessory Equalizer Settings menu.

**Table 3-194** `RetEQIndexName` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)



Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x08	Lingo ID: Accessory Equalizer lingo
4	0x07	Command ID: RetEQIndexName
5	0xNN	eqIndex: the selected accessory Equalizer Index
6-NN	0xNN...	The accessory equalizer name, as a null-terminated UTF-8 character array
(last byte)	0xNN	Checksum

## Lingo 0x09: Sports Lingo

The Sports lingo enables gym cardio equipment to communicate with an iPod to track a user's workout while the user continues to use the iPod for other purposes. The Sports lingo version of the iPod must be 1.01 or later to support these accessories. For information about determining the lingo version, see [“Determining iPod Support for Cardio Equipment”](#) (page 495).

All Sports lingo commands require authentication level 2.0.

### Sports Lingo commands

[Table 3-195](#) (page 277) summarizes the Sports commands (lingo 0x09).

**Table 3-195** Sports lingo command summary

Command	ID	Direction	Data length	Protocol version
DeviceACK	0x00	Dev to iPod	0x02	1.01
GetDeviceVersion	0x01	iPod to Dev	0x00	1.01
RetDeviceVersion	0x02	Dev to iPod	0x02	1.01
GetDeviceCaps	0x03	iPod to Dev	0x00	1.01
RetDeviceCaps	0x04	Dev to iPod	0x03	1.01
Reserved	0x05-0x7F	N/A		
iPodACK	0x80	iPod to Dev	0x02	1.01
Reserved	0x81-0x82	N/A		
Get iPodCaps	0x83	Dev to iPod	0x00	1.01

Command	ID	Direction	Data length	Protocol version
RetiPodCaps	0x84	iPod to Dev	0x03	1.01
GetUserIndex	0x85	Dev to iPod	0x00	1.01
RetUserIndex	0x86	iPod to Dev	0x01	1.01
Reserved	0x87	N/A		
GetUserData	0x88	Dev to iPod	0x01	1.01
RetUserData	0x89	iPod to Dev	0xNN	1.01
SetUserData	0x8A	Dev to iPod	0xNN	1.01
Reserved	0x8B-0xFF	N/A		

## Command History of the Sports Lingo

Table 3-196 (page 278) shows the history of changes to the Sports lingo.

**Table 3-196** Sports lingo command history

Lingo version	Command changes	Features
1.01	Add: 0x00–0x04, 0x80, 0x83–0x86, 0x88–0x8A	Cardio equipment accessory support

## Command 0x00: DeviceACK

Direction: Device to iPod

This command is sent by the device in response to a command sent from the iPod. An ACK response must be sent for any of the following conditions:

- A bad parameter is received
- An unsupported/invalid command is received
- A command that does not return any data has completed

**Table 3-197** DeviceACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x04	Length of Packet

Byte number	Value	Comment
3	0x09	Lingo ID: Sports
4	0x00	Command ID: DeviceACK
5	0xNN	ackStatus of command; see Table 3-198 (page 279) for details
6	0xNN	ID of command being acknowledged
7	0xNN	Checksum

Table 3-198 (page 279) presents the possible values for the `ackStatus` byte.

**Table 3-198** `ackStatus` details

ackStatus	Description
0x00	Success
0x01	N/A (reserved)
0x02	Command failed
0x03	Out of resources
0x04	Bad parameter
0x05	N/A (reserved)
0x06	N/A (reserved)
0x07	Not authenticated
0x08-0xFF	Reserved

## Command 0x01: GetDeviceVersion

Direction: iPod to Device

This command is sent by iPod to obtain the sports device lingo protocol version. In response, the device will send the `RetDeviceVersion` command with its major and minor protocol version numbers. Devices may query the iPod's Sports Lingo protocol version using the General Lingo `RequestLingoProtocolVersion` command.

**Table 3-199** `GetDeviceVersion` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)

Byte number	Value	Comment
2	0x02	Length of Packet
3	0x09	Lingo ID: Sports
4	0x01	Command ID: <code>GetDeviceVersion</code>
5	0xF4	Checksum

## Command 0x02: `RetDeviceVersion`

---

Direction: Device to iPod

This command must be returned by the device in response to each `GetDeviceVersion` command received from the iPod. The device returns the maximum lingo version that it supports. The first byte is the major version number (tens/ones digits left of the decimal point) and the second byte is the minor version number (tenths/hundreths digits right of the decimal point). The iPod will support all device lingo versions up to and including its own current lingo version.

**Table 3-200** `RetDeviceVersion` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x04	Length of Packet
3	0x09	Lingo ID: Sports
4	0x02	Command ID: <code>RetDeviceVersion</code>
5	0xNN	Major Sports lingo protocol version supported by device (currently 0x01)
6	0xNN	Minor Sports lingo protocol version supported by device (currently 0x01)
7	0xNN	Checksum

## Command 0x03: `GetDeviceCaps`

---

Direction: iPod to Device

This command is sent by the iPod to get the sports device capabilities and determine the features present on the device. In response, the device will send the `GetDeviceCaps` command with the payload indicating the capabilities supported.

**Table 3-201** GetDeviceCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x02	Length of Packet
3	0x09	Lingo ID: Sports
4	0x03	Command ID: GetDeviceCaps
5	0xF2	Checksum

## Command 0x04: RetDeviceCaps

---

Direction: Device to iPod

This command must be sent by the device in response to each `GetDeviceCaps` command sent by the iPod. The sports device returns the payload indicating which of the capabilities it supports. When a capability bit is set, it means that the associated command is supported by the device. Conversely, if a capability bit is clear, it means that the associated commands are not supported by the device.

**Table 3-202** RetDeviceCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x05	Length of Packet
3	0x09	Lingo ID: Sports
4	0x04	Command ID: GetDeviceCaps
5	0xNN	capsMask (bits15:8)
6	0x00	capsMask (bits7:0)
7	0x00	Reserved (set equal to 0x00)
8	0xNN	Checksum

[Table 3-203](#) (page 282) presents the possible values for the `capsMask` bits.

**Table 3-203** RetDeviceCaps capsMask details

capsMask	Description
bits 0-8	Reserved (set to 0)
bit 9	Accessory supports (1) or does not support (0) Sports lingo commands 0x80 through 0x8A
bits 10-15	Reserved (set to 0)

## Command 0x80: iPodACK

Direction: iPod to Device

This command is sent by the iPod in response to a command sent from the device. An ACK response is sent when a bad parameter is received, an unsupported/invalid command is received, or a command that does not return any data has completed.

**Table 3-204** iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x04	Length of Packet
3	0x09	Lingo ID: Sports
4	0x80	Command ID: iPodACK
5	0xNN	ackStatus of Command
6	0xNN	ID of Command being acknowledged
7	0xNN	Checksum

[Table 3-205](#) (page 282) presents the possible values for the iPodACK byte.

**Table 3-205** iPodACK ackStatus details

ackStatus	Description
0x00	Success
0x01	N/A (reserved)
0x02	Command failed
0x03	Out of resources
0x04	Bad parameter

ackStatus	Description
0x05-0xFF	N/A (reserved)

## Command 0x83: GetiPodCaps

Direction: Device to iPod

This command may be sent by the device to get the iPod capabilities and determine if required features are present. In response, the iPod will send the `RetiPodCaps` command with the payload indicating the capabilities supported.

**Note:** The `GetiPodCaps` command must be sent before the device may use any commands that rely upon iPod support for gym cardio equipment features, such as `GetUserIndex`.

**Table 3-206** GetiPodCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x02	Length of Packet
3	0x09	Lingo ID: Sports
4	0x83	Command ID: GetiPodCaps
5	0x72	Checksum

## Command 0x84: RetiPodCaps

Direction: iPod to Device

This command is sent by the iPod in response to the `GetiPodCaps` command sent by the device. The iPod returns the payload indicating which of the capabilities it supports. The user count is valid only if the iPod user data capability bit is set.

**Table 3-207** RetiPodCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x05	Length of Packet

Byte number	Value	Comment
3	0x09	Lingo ID: Sports
4	0x84	Command ID: RetiPodCaps
5	0x00	capsMask (bits15:8)
6	0xNN	capsMask (bits7:0)
7	0xNN	userCount number of user data profiles on iPod
8	0xNN	Checksum

Table 3-208 (page 284) presents the possible values for the capsMask bits.

**Table 3-208** RetiPodCaps capsMask details

capsMask	Description
bit 0	Cardio equipment is supported (1) or is not supported (0).
bit 1	User data is supported (1) or is not supported (0). The userCount value is valid if and only if this capability bit is set.
bit2-15	Reserved (set to 0)

## Command 0x85: GetUserIndex

Direction: Device to iPod

This command may be sent by the device to obtain the current user index selection from the iPod. In response, the iPod will return the RetUserIndex command with the current user index.

**Note:** This command must not be sent unless the device has already sent a GetiPodCaps command and received a RetiPodCaps command indicating user data support capability. If the iPod does not support user data, it will return an iPodACK with a bad parameter status.

**Table 3-209** GetUserIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x02	Length of Packet
3	0x09	Lingo ID: Sports
4	0x85	Command ID: GetUserIndex



Byte number	Value	Comment
5	0x70	Checksum

## Command 0x86: RetUserIndex

Direction: iPod to Device

This command is sent by the iPod in response to the `GetUserIndex` command received from the device. The current user index is returned (the first user has index 0). The total user count present on the iPod is obtained using the `GetiPodCaps` command.

**Table 3-210** RetUserIndex packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x03	Length of Packet
3	0x09	Lingo ID: Sports
4	0x86	Command ID: RetUserIndex
5	0xNN	userIndex (0 to userCount-1) (see Note below)
6	0xNN	Checksum

**Note:** The `userIndex` value 0 is reserved by iPod for the default/unknown user profile. An iPod that does not support multiple user profiles but does support user data will offer only `userIndex` 0.

## Command 0x88: GetUserData

Direction: Device to iPod

This command may be sent by the device to obtain the current user's data. In response, the iPod will return the `RetUserData` command with the specified `userDataType`. If the iPod does not support this command or the `userDataType` is not valid, iPod will return an `iPodACK` with a bad parameter status.

**Note:** This command must not be sent unless the device has already sent a `Get iPodCaps` command and received a `Ret iPodCaps` command indicating user data support capability. If the iPod does not support user data, it will return an `iPodACK` with a bad parameter status.

**Table 3-211** `GetUserData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x03	Length of Packet
3	0x09	Lingo ID: Sports
4	0x88	Command ID: <code>GetUserData</code>
5	0xNN	<code>userDataType</code>
6	0xNN	Checksum

[Table 3-212](#) (page 286) presents the possible values for the `userDataType` byte.

**Table 3-212** `GetUserData` `userDataType` details

<code>userDataType</code>	Description
0x00	Preferred Unit System
0x01	Name
0x02	Gender
0x03	Weight
0x04	Age
0x05	Workout Recording Preference
0x06-0xFF	Reserved

## Command 0x89: `RetUserData`

Direction: iPod to Device

This command is sent by the iPod in response to the `GetUserData` command received from the device. The iPod returns the requested user data for the specified data type.

**Table 3-213** RetUserData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0xNN	Length of Packet
3	0x09	Lingo ID: Sports
4	0x89	Command ID: RetUserData
5	0xNN	userDataType
6...N	0xNN	userData (variable length)
(last byte)	0xNN	Checksum

[Table 3-214](#) (page 287) presents the possible values for the userDataType byte and userData field values.

**Table 3-214** RetUserData userDataType details

userDataType	userData
0x00	Preferred unit system (1 byte): 0x00 = No information 0x01 = Imperial units (lbs/miles) 0x02 = Metric units (kg/km) 0x03-0xFF = Reserved
0x01	Name (variable length, 128 bytes max, null-terminated UTF-8 string)
0x02	Gender (1 byte): 0x00 = No information 0x01 = Female 0x02 = Male 0x03-0xFF = Reserved
0x03	Weight in tenths of a kg (2 bytes, MSB first): 0x0000 = No information 0x0001 = 0.1 kg 0x0200 = 51.2 kg 0x1388 = 500.0 kg (maximum weight limit) 0x1389-0xFFFF = Reserved

userDataType	userData
0x04	Age in years (1 byte): 0x00 = No information 0x20 = 32 years old 0xC8 = 200 years old (maximum age limit) 0xC9-0xFF = Reserved
0x05	Workout recording preference (1 byte): 0x00 = No information 0x01 = Never record workout data 0x02 = Ask if workout should be recorded 0x03 = Always record workout data 0x04-0xFF = Reserved
0x06-0xFF	Reserved

## Command 0x8A: SetUserData

This command may be sent by the device to set the current user's data. In response, the iPod will send the `iPodACK` command with the status of the operation. If the iPod does not support this command or the `userDataType` or `userData` is not valid, an `iPodACK` with a bad parameter status is returned.

**Note:** This command must not be sent unless the device has already sent a `GetiPodCaps` command and received a `RetiPodCaps` command indicating user data support capability. If the iPod does not support user data, it will return an `iPodACK` with a bad parameter status.

**Table 3-215** SetUserData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0xNN	Length of Packet
3	0x09	Lingo ID: Sports
4	0x8A	Command ID: SetUserData
5	0xNN	userDataType
6...N	0xNN	userData (variable length)
(last byte)	0xNN	Checksum

Table 3-216 (page 289) presents the possible values for the `userDataType` byte and `userData` field values.

**Table 3-216** SetUserData `userDataType` details

<code>userDataType</code>	<code>userData</code>
0x00	Preferred unit system (1 byte): 0x00 = No information 0x01 = Imperial units (lbs/miles) 0x02 = Metric units (kg/km) 0x03-0xFF = Reserved
0x01	Setting not allowed
0x02	Gender (1 byte): 0x00 = No information 0x01 = Female 0x02 = Male 0x03-0xFF = Reserved
0x03	Weight in tenths of a kg (2 bytes, MSB first): 0x0000 = No information 0x0001 = 0.1 kg 0x0200 = 51.2 kg 0x1388 = 500.0 kg (maximum weight limit) 0x1389-0xFFFF = Reserved
0x04	Age in years (1 byte): 0x00 = No information 0x20 = 32 years old 0xC8 = 200 years old (maximum age limit) 0xC9-0xFF = Reserved
0x05	Setting not allowed
0x06-0xFF	Reserved

## Lingo 0x0A: Digital Audio Lingo

The Digital Audio lingo supports iAP commands, using either USB or UART transport, that let the iPod transfer digital audio to an accessory. The iPod uses these lingo commands to retrieve a list of supported sample rates from the accessory and to inform the accessory of the iPod's current sample rate, Sound Check value, and track volume adjustment value (which is set using iTunes). After these interchanges, the iPod performs

audio sample rate conversion internally and transfers digital audio to the accessory device at one of the device's supported audio data sample rates. For a sample command sequence that declares this lingo, see [Table 3-273](#) (page 337).

**Note:** The iPod models that contain version 1.00 of the Digital Audio lingo do not correctly support digital audio. An accessory should check the attached iPod's version of the Digital Audio lingo and use digital audio only if the version number is greater than 1.00. See [Table 1-2](#) (page 36) and [Table 1-3](#) (page 37) for lists of affected iPod models.

## Accessory Authentication

Every accessory device that supports the Digital Audio lingo must authenticate itself with a connected iPod as soon as the iPod recognizes the device; deferred authentication is not permitted. This means that the accessory's IDPS process must specify authentication immediately after identification in its `IdentifyToken` (see [Table 2-72](#) (page 112)). With iPods that do not support IDPS, the accessory must send an `IdentifyDeviceLingoes` command with the authentication control bits in its `options` field set to `10b` (Authenticate immediately after identification). See [Table 2-34](#) (page 83) for details.

**Note:** The General lingo `Identify` command cannot be used to identify an accessory device for any purpose if the device supports the Digital Audio lingo.

When the accessory identifies itself as supporting the Digital Audio lingo, authentication can happen in the background and the iPod can proceed to transfer digital audio as if authentication were successful.

The Digital Audio lingo ID is 0x0A. All its commands are transferred in small packet format and all require authentication. They are listed in [Table 3-217](#) (page 290).

**Table 3-217** Digital Audio lingo command summary

ID	Command	Data length	Protocol version	Authentication required
0x00	<code>AccAck</code>	0x04	1.00	Yes
0x01	<code>iPodAck</code>	0x04	1.00	Yes
0x02	<code>GetAccSampleRateCaps</code>	0x02	1.00	Yes
0x03	<code>RetAccSampleRateCaps</code>	<i>NN</i>	1.00	Yes
0x04	<code>NewiPodTrackInfo</code>	0x0E	1.00	Yes
0x05	<code>SetVideoDelay</code>	0x06	1.03	Yes
0x06–0xFF	Reserved	N/A	N/A	N/A

## USB Audio Transport

---

Digital audio on the iPod supports USB Audio 1.0, with the addition that it will transfer digital audio on a High-Speed USB 2.0 bus as well as on a Full-Speed USB 1.1 bus. During transport, the accessory is the USB host, the iPod is the USB device, and PCM data is synchronized with the 1 ms USB start-of-frame (SOF) packets. More information about USB and its standards is available at <http://www.usb.org>.

**Note:** The iPod USB isochronous audio data endpoint descriptor `bmAttributes` field erroneously returns the Synchronization Type field (D3:2) as b00 (no synchronization) instead of the correct value, b11 (synchronous). The iPod supports synchronous data transfers, so USB host devices must override these attribute bits. The erroneous b00 value is retained for backwards compatibility with older iPod accessories.

**Note:** Any accessory that outputs digital audio obtained from an iPod must implement copy protection in its output stream; for example, by setting Serial Copy Management System (SCMS) bits to 10.

When receiving digital audio over a USB audio streaming interface, the accessory may buffer data to achieve consistent audio playback. Since buffering introduces latency, it is suggested that this latency be kept below 200 ms to ensure timely response to user input. Apple may enforce this suggestion in the future.

### Digital Audio and Extended Interface Mode

---

Digital Audio lingo version 1.01 requires the iPod to be in Extended Interface mode before USB audio can be transferred. With versions 1.02 and later, USB audio can be transferred in any mode.

### Audio/Video Synchronization for Digital Audio

---

Digital Audio lingo versions 1.03 and above allow synchronization between iPod video and the sound for that video that is being transmitted as digital audio. The video may either appear on the iPod's display or be transmitted through the analog video output. For synchronization to occur, the accessory must use the `SetVideoDelay` command to send a fixed time (in milliseconds) that the iPod will add as a delay to its own video. The delay should be equal to the audio latency of the accessory; it corresponds to the difference between the time that digital audio data appears on the digital audio transport (USB) and the time the corresponding analog audio is sent to the speakers. The accessory must resend the `SetVideoDelay` command whenever its audio latency changes, for example at sample rate changes.

### Line Level Output and Digital Audio Transport

---

Line level output and digital audio transport are mutually exclusive. All iPod audio is sent to the LINE-OUT pins of the 30-pin connector until digital audio transport is enabled, at which point the iPod's audio output is redirected to the USB audio streaming interface. When digital audio transport is disabled, iPod audio is again sent to the LINE-OUT pins (see [Table 3-1](#) (page 145)).

When disabling digital audio transport, the iPod does not automatically enter the playback Pause state, but instead continues in its current Play or Pause state. The only difference is that the output mode changes to LINE-OUT. If appropriate, the accessory should set the iPod to Pause when disabling digital audio.

## Enabling and Disabling USB Audio

---

Digital audio is enabled and LINE-OUT audio is disabled when:

- Version 1.01: The accessory selects a nonzero-bandwidth alternate USB audio streaming interface and enters Extended Interface mode.
- Version 1.02: The accessory replies to a `GetAccSampleRateCaps` command with a list of valid sample rates.

Digital audio is disabled and LINE-OUT audio is enabled when:

- Version 1.01: The accessory selects a zero-bandwidth alternate USB audio streaming interface, reidentifies itself without supporting the Digital Audio lingo, or disconnects from the USB Audio Configuration (that is, it disconnects the USB cable).
- Version 1.02: The accessory either reidentifies itself without supporting the Digital Audio lingo or disconnects from the USB Audio Configuration (that is, it disconnects the USB cable).

To enable USB audio, an iPod and its attached accessory must perform the following steps, in the order listed, after the user has connected the iPod to the accessory:

1. The iPod's USB enumeration lists two configurations: 1 = Mass Storage, 2 = USB Audio and HID device.
2. The device sends the USB standard request `Set_Configuration` to select configuration 2.
3. The accessory completes the IDPS identification process, declaring the General lingo and the Digital Audio lingo, plus the Extended Interface lingo if the iPod does not support Digital Audio lingo version 1.02. With iPods that do not support IDPS, the `IdentifyDeviceLingoes` command may be used. The iPod authenticates the device for the requested lingoes.
4. The device selects a nonzero-bandwidth alternate USB audio streaming interface on the iPod.
5. The device tells the iPod to enter Extended Interface Mode using the General lingo `EnterRemoteUIMode` command (not required for Digital Audio lingo version 1.02). If an audio track is playing, playback will pause. If a video track is playing, playback will stop.
6. The device places the iPod in the Play state.
7. The iPod sends a `NewiPodTrackInfo` command to the device.
8. The device acknowledges the `NewiPodTrackInfo` command using the `AccAck` command.
9. The device configures its playback DAC to the iPod's sample rate.
10. The device sends a USB audio `SET_CUR` request for the `SAMPLING_FREQ_CONTROL` control, passing a sample rate equal to the value sent by the `NewiPodTrackInfo` command.
11. The device requests digital audio packets from the isochronous USB pipe.



**Note:** Step 2 is slightly different if a 191 k $\Omega$  resistor is used as the identify resistor (see “Accessory Detect and Identify” in *iPod/iPhone Hardware Specifications*). In this case, the iPod will present the USB audio and HID device configuration as the first configuration, and the accessory will be required to select configuration 1 through the `Set_Configuration` USB standard request.

An example of the Digital Audio identification process is shown in [Table 3-273](#) (page 337).

**Note:** With Digital Audio lingo version 1.01, switching from the zero bandwidth alternate USB audio streaming interface to the nonzero bandwidth interface without first exiting and entering Extended Interface Mode can prevent a `NewiPodTrackInfo` command from being sent from the iPod when a track change occurs. The accessory must not select the zero-bandwidth alternate interface on the iPod, even when changing sample rates, unless it is also exiting Extended Interface Mode.

With Digital Audio lingo version 1.01, reenabling digital audio after an accessory has disabled it requires the following actions:

1. Enable the nonzero-bandwidth alternate USB audio streaming interface on the iPod.
2. Enter Extended Interface Mode using the General lingo `EnterRemoteUIMode`. This will pause iPod playback.
3. Place the iPod in the Play state.
4. Receive a `NewiPodTrackInfo` command from the iPod.
5. Acknowledge the `NewiPodTrackInfo` command using the `AccAck` command.
6. Configure the accessory’s playback DAC to the iPod’s sample rate.
7. Send a USB audio `SET_CUR` request for the `SAMPLING_FREQ_CONTROL` control, passing a sample rate equal to the value sent by the `NewiPodTrackInfo` command.
8. Request digital audio packets from the isochronous USB pipe.

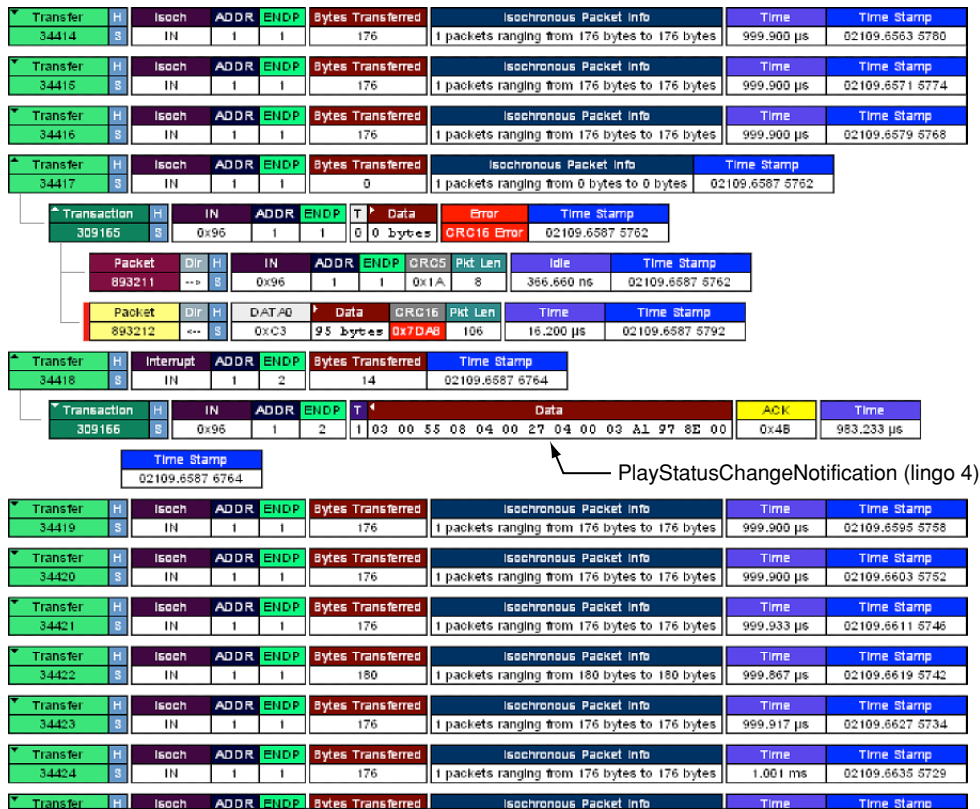
If the accessory requests digital audio data from the isochronous USB pipe before digital audio is enabled or before the correct digital sample rate has been negotiated, the iPod will return isochronous packets filled with zeros. The iPod will also return packets filled with zeros if authentication fails.

## USB Audio Errors on Older iPods

On first-generation nano and 5G iPods, USB Audio data from the iPod will occasionally contain 16-bit cyclic redundancy check (CRC16) errors. If iAP commands are sent from the iPod to a USB host while USB audio is enabled, there is a possibility that the USB audio `DATA0` packet immediately preceding the iAP command will contain a CRC16 error. This error occurs infrequently; however, enabling iPod notifications over USB (such as the Lingo `0x04 PlayStatusChangeNotification` command, which is sent every 500 ms when enabled) greatly increases the possibility that a CRC16 error will occur.

To resolve this problem, the USB host must be able to recover immediately from a CRC16 error in the payload of the isochronous USB audio data. [Figure 3-1](#) (page 294) illustrates a typical recovery sequence.

Figure 3-1 A USB host recovers from a CRC16 error



## Command History of the Digital Audio Lingo

Table 3-218 (page 294) shows the history of command changes in the Digital Audio lingo:

Table 3-218 Digital Audio lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x04	Digital audio sample rate and track information support
1.01	None	The <code>NewiPodTrackInfo</code> command is resent until it is acknowledged by the USB host with an <code>AccAck</code> command. Also corrected a bug where <code>NewiPodTrackInfo</code> was not being sent before every track; for this reason, accessories should not use Digital Audio with iPods that support only Digital Audio lingo version 1.00.
1.02	None	The Digital Audio lingo no longer requires the iPod to be in Extended Interface mode.
1.03	Add 0x05	Added <code>SetVideoDelay</code> to allow digital audio to synchronize with video playback.

## Command 0x00: AccAck

---

Direction: Device to iPod

The accessory sends the `AccAck` command to acknowledge the receipt of a command from the iPod and returns the command status (see [Table 3-220](#) (page 295)). An `AccAck` command should be sent only for commands whose documentation specifies that an `AccAck` command is needed.

**Table 3-219** `AccAck` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x04	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x00	Command ID: <code>AccAck</code>
5	0xNN	Command status. See <a href="#">Table 3-220</a> (page 295)
6	0xNN	The ID for the command being acknowledged
7	0xNN	Checksum

**Table 3-220** `AccAck` status values

status	Meaning
0x00	Success (OK)
0x01	Reserved
0x02	ERROR: Command failed
0x03	ERROR: Out of resources
0x04	ERROR: Bad parameter
0x05	ERROR: Unknown ID
0x06	Reserved
0x07	ERROR: Accessory not authenticated
0x08–0xFF	Reserved

## Command 0x01: iPodAck

Direction: iPod to Device

The iPod sends the `iPodAck` command when it receives an invalid or unsupported command or a bad parameter. The command status returns are the same as those used for the `AccAck` command (see [Table 3-220](#) (page 295)).

**Table 3-221** `iPodAck` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x04	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x01	Command ID: <code>iPodAck</code>
5	0xNN	Command status. See <a href="#">Table 3-220</a> (page 295)
6	0xNN	The ID for the command being acknowledged
7	0xNN	Checksum

## Command 0x02: GetAccSampleRateCaps

Direction: iPod to Device

The iPod uses this command to request the list of supported sample rates from the accessory. The iPod sends it after the accessory indicates its support of the Digital Audio lingo during its identification process. The accessory responds to this command using the `RetAccSampleRateCaps` command.

After the iPod sends a `GetAccSampleRateCaps` command, it waits up to 3 seconds for a `RetAccSampleRateCaps` command from the accessory before timing out. If a timeout occurs, the iPod resends the command. It will resend the command up to three times if necessary, and if all three attempts fail, the iPod will then respond to any digital audio request with zeros.

If the iPod receives a `RetAccSampleRateCaps` command with invalid parameters, it sends the appropriate command status using an `iPodAck` command (see [Table 3-220](#) (page 295)). At this point the accessory should send a `RetAccSampleRateCaps` with correct parameters. The iPod allows up to three retries if necessary, and if all three attempts fail, the iPod then responds to any digital audio request with zeros.

**Table 3-222** `GetAccSampleRateCaps` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)

Byte number	Value	Comment
1	0x55	Start of packet
2	0x02	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x02	Command ID: GetAccSampleRateCaps
5	0xF2	Checksum

## Command 0x03: RetAccSampleRateCaps

Direction: Device to iPod

An accessory sends the `RetAccSampleRateCaps` command in response to a `GetAccSampleRateCaps` command from an iPod.

The accessory returns the list of sample rates it supports with this command. The sample rates must be taken from the list of iPod supported sample rates shown in [Table 3-224](#) (page 298). Any audio encoded at an unsupported sample rate is resampled internally by the iPod to conform to a supported sample rate. In general, audio quality will be better when no resampling is performed; therefore accessories should support as many values listed in [Table 3-224](#) (page 298) as possible.

**Note:** At a minimum, every accessory must support the sample rates 32 KHz, 44.1 KHz, and 48 KHz.

A `RetAccSampleRateCaps` command with sample rates not listed in [Table 3-224](#) (page 298), or missing any of the required sample rates, is invalid. If the iPod receives such a command, it sends the accessory an `iPodAck` command with a negative acknowledgment as the command status.

**Table 3-223** `RetAccSampleRateCaps` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	2+4n	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x03	Command ID: <code>RetAccSampleRateCaps</code>
5 ... (5+4n-1)	0xNNNNNNNN	A list of $n$ sample rates (32-bit big-endian format) supported by the accessory. The sample rates must be taken from <a href="#">Table 3-224</a> (page 298).
5+4n	0xNN	Checksum

**Table 3-224** Digital audio sample rates supported by iPods (in Hertz)

Decimal	Hexadecimal	Required/Optional
8000	0x00001F40	Optional
11025	0x00002B11	Optional
12000	0x00002EE0	Optional
16000	0x00003E80	Optional
22050	0x00005622	Optional
24000	0x00005DC0	Optional
32000	0x00007D00	Optional
44100	0x0000AC44	Required
48000	0x0000BB80	Required

## Command 0x04: NewiPodTrackInfo

---

Direction: iPod to Device

The iPod sends the `NewiPodTrackInfo` command before the first audio track begins playing. It sends the command again whenever it starts playing a track with different sample rate, sound check, or track volume parameters. In response to this command, accessories should prepare themselves to receive audio data with the new parameters.

The accessory must acknowledge this command, using the `AccAck` command, but the iPod does not wait for this acknowledgment before allowing digital audio to be transferred to the accessory.

The sample rate sent to the accessory is taken from the list of sample rates returned to the iPod by the `RetAccSampleRateCaps` command. If the accessory supports the sample rate of the current audio track, then it is sent as the current sample rate. If the accessory does not support the sample rate, the iPod resamples the audio data to a supported sample rate in real time and sends this new supported sample rate as the current sample rate.

The Sound Check value and track volume adjustment value are the corresponding values set by iTunes, rounded to the nearest integer. The values represent gain (in decibels) and may be positive or negative. Note that if the Sound Check option in the iPod is disabled, the `NewiPodTrackInfo` command always sends 0 as the new Sound Check value.

When the iPod sends a `NewiPodTrackInfo` command, it waits up to 500 ms for the `AccAck` command before timing out. If a timeout occurs or if the `AccAck` returns an error as the command status, the iPod sends the command again.

**IMPORTANT:** The `NewiPodTrackInfo` command should not be used as a general mechanism to detect track changes. Use appropriate commands from the Display Remote or Extended Interface lingo instead.

**Table 3-225** `NewiPodTrackInfo` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x0E	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x04	Command ID: <code>NewiPodTrackInfo</code>
5	0xNN	New sample rate, bits 31:24
6	0xNN	New sample rate, bits 23:16
7	0xNN	New sample rate, bits 15:8
8	0xNN	New sample rate, bits 7:0
9	0xNN	New Sound Check value, bits 31:24
10	0xNN	New Sound Check value, bits 23:16
11	0xNN	New Sound Check value, bits 15:8
12	0xNN	New Sound Check value, bits 7:0
13	0xNN	New track volume adjustment, bits 31:24
14	0xNN	New track volume adjustment, bits 23:16
15	0xNN	New track volume adjustment, bits 15:8
16	0xNN	New track volume adjustment, bits 7:0
17	0xNN	Checksum

## Command 0x05: `SetVideoDelay`

Direction: Device to iPod

The device sends this command to inform the iPod of a new delay (in milliseconds) that must be applied to iPod video to synchronize it with the audio for the currently playing video. See ["Audio/Video Synchronization for Digital Audio"](#) (page 291). The iPod responds to `SetVideoDelay` with an ["iPodAck"](#) (page 296) command.

**Table 3-226** SetVideoDelay packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART transport)
1	0x55	Start of packet
2	0x06	Length of packet payload
3	0x0A	Lingo ID: Digital audio lingo
4	0x05	Command ID: SetVideoDelay
5	0xNN	Video delay in milliseconds, bits 31:24
6	0xNN	Video delay in milliseconds, bits 23:16
7	0xNN	Video delay in milliseconds, bits 15:8
8	0xNN	Video delay in milliseconds, bits 7:0
9	0xNN	Checksum

## Lingo 0x0C: Storage Lingo

The iAP Storage lingo, Lingo 0x0C, lets an accessory device store files on an attached iPod as if it were a hard drive. Use of the Storage lingo requires accessory authentication.

### Command History of the Storage Lingo

Table 3-227 (page 300) shows the history of command changes in the Storage lingo:

**Table 3-227** Storage lingo command history

Lingo version	Command changes	Features
1.01	Add: 0x00-0x02, 0x04, 0x07-08, 0x10-0x12	Support for iTunes Tagging.
1.02	Add: 0x80-0x82. Add options to 0x12	Add support for Sports lingo.

### Command Summary

Table 3-228 (page 301) lists the Storage lingo commands.



**Table 3-228** Storage lingo commands

CmdID	Name	Direction	Payload:bytes
0x00	iPodACK	iPod to Dev	[ackStatus:1, cmdID:1, handle:1, (transactionID:2)]
0x01	GetiPodCaps	Dev to iPod	[None]
0x02	RetiPodCaps	iPod to Dev	[totalSpace:8, maxFileSize:4, maxWriteSize:2, Reserved:6, majorVersion:1, minorVersion:1]
0x03	Reserved		
0x04	RetiPodFileHandle	iPod to Dev	[handle:1]
0x05-0x06	Reserved		
0x07	WriteiPodFileData	Dev to iPod	[offset:4, handle:1, data:<var>]
0x08	CloseiPodFile	Dev to iPod	[handle:1]
0x09-0x0F	Reserved		
0x10	GetiPodFreeSpace	Dev to iPod	[None]
0x11	RetiPodFreeSpace	iPod to Dev	[freeSpace:8]
0x12	OpeniPodFeatureFile	Dev to iPod	[feature: 0xNN]
0x13-0x7F	Reserved		
0x80	DeviceACK	Dev to iPod	[ackStatus:1, cmdID:1, handle:1]
0x81	GetDeviceCaps	iPod to Dev	[feature: 0xNN]
0x82	RetDeviceCaps	Dev to iPod	[feature: 0xNN]
0x83-0xFF	Reserved		

The following is the typical sequence of commands used to store data on an iPod:

1. Complete the device identification and authentication processes, as specified in ["Device Signaling and Initialization"](#) (page 48), making sure to identify the accessory as supporting the Storage lingo.
2. Send `GetiPodCaps` and receive `RetiPodCaps`, which returns the maximum write size.
3. Send `OpeniPodFeatureFile` and receive `RetiPodFileHandle`. The returned file handle points to a specific area in the iPod's file system. Currently the only accessible area is `/iPod_Control/Device/Accessories/Tags/`, used by the Radio Tagging System (see ["iTunes Tagging"](#) (page 463)).
4. Send one or more `WriteiPodFileData` commands with the data to be stored. Receive an `iPodACK` command for each one.

5. Send `CloseiPodFile` when finished writing all the data. Alternately, all files are automatically closed when the accessory detaches.

**Note:** The accessory should use "Command 0x10: GetiPodFreeSpace" (page 307) to verify that there is adequate free space available on the iPod's drive before attempting to open or write data to a file. If there is less than 5 MB available, then calls to `OpeniPodFeatureFile` or `WriteiPodFileData` may fail for lack of memory. The recommended practice is to check the free space before opening or writing data, and to handle any out-of-memory ACK commands by retaining the data in internal storage and displaying an "iPod Full" message on the accessory's display.

## Command 0x00: iPodACK

Direction: iPod to Device

The iPod sends this command to acknowledge the receipt of a Storage lingo command from the accessory.

**Table 3-229** Storage iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x00	Command ID: iPodACK
5	0xNN	Command result status. See Table 3-230 (page 302).
6	0xNN	The ID for the command being acknowledged.
7	0xNN	The file handle for the command (0xFF if not applicable.)
8	0xNN	Checksum

**Table 3-230** iPodACK responses

Value	Description
0x00	Success
0x01	N/A (reserved)
0x02	Command failed
0x03	Out of resources
0x04	Bad parameter

Value	Description
0x05	Unknown ID
0x06	N/A (reserved)
0x07	Not authenticated
0x08	Bad authentication version
0x09	N/A (reserved)
0x0A	N/A (reserved)
0x0B	N/A (reserved)
0x0C	N/A (reserved)
0x0D	Invalid file handle
0x0E-0xFF	Reserved

## Command 0x01: GetiPodCaps

---

Direction: Device to iPod

The accessory asks the iPod to return its storage capabilities. The iPod replies with `RetiPodCaps`.

**Table 3-231** `GetiPodCaps` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x01	Command ID: <code>GetiPodCaps</code>
5	0xF1	Checksum

## Command 0x02: RetiPodCaps

---

Direction: iPod to Device

The iPod tells the accessory about the iPod's storage capabilities:

- `totalSpace` is the amount of storage on the iPod in bytes, including space currently in use.

- `maxFileSize` is the largest possible size, in bytes, of any file on the iPod.
- `maxWriteSize` is the largest amount of data, in bytes, that can be written to the iPod in a single `WriteiPodFileData` command.
- `majorVersion` and `minorVersion` are the version number of the Storage lingo protocol implemented by the device.

**Table 3-232** RetiPodCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x18	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x02	Command ID: RetiPodCaps
5	0xNN	<code>totalSpace</code> (bits 63:56). The total amount of storage space on the iPod in bytes, including space currently in use.
6	0xNN	<code>totalSpace</code> (bits 55:48)
7	0xNN	<code>totalSpace</code> (bits 47:40)
8	0xNN	<code>totalSpace</code> (bits 39:32)
9	0xNN	<code>totalSpace</code> (bits 31:24)
10	0xNN	<code>totalSpace</code> (bits 23:16)
11	0xNN	<code>totalSpace</code> (bits 15:8)
12	0xNN	<code>totalSpace</code> (bits 7:0)
13	0xNN	<code>maxFileSize</code> (bits 31:24). The size in bytes of the largest possible file on the iPod.
14	0xNN	<code>maxFileSize</code> (bits 23:16)
15	0xNN	<code>maxFileSize</code> (bits 15:8)
16	0xNN	<code>maxFileSize</code> (bits 7:0)
17	0xNN	<code>maxWriteSize</code> (bits 15:8). The size in bytes of the largest possible amount of data than can be sent to the iPod with <code>WriteiPodFile</code> .
18	0xNN	<code>maxWriteSize</code> (bits 7:0)
19	0xNN	Reserved.
20	0xNN	Reserved.

Byte number	Value	Comment
21	0xNN	Reserved.
22	0xNN	Reserved.
23	0xNN	Reserved.
24	0xNN	Reserved.
25	0xNN	majorVersion (1 byte). The major version number.
26	0xNN	minorVersion (1 byte). The minor version number.
27	0xNN	Checksum

## Command 0x04: RetiPodFileHandle

---

Direction: iPod to Device

The iPod returns a unique 8-bit handle to identify the file.

The value of `handle` is a session-specific identifier that represents a file. This is similar to a Unix file descriptor. The handle is valid until either the accessory is detached or the accessory sends a `CloseiPodFile` command with this handle.

**Table 3-233** RetiPodFileHandle packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x04	Command ID: RetiPodFileHandle
5	0xNN	handle (1 byte)
6	0xNN	Checksum

## Command 0x07: WriteiPodFileData

---

Direction: Device to iPod

The accessory writes a block of data to a file starting at the offset passed with this command. The file must have been previously opened for writing; failure is reported as a bad parameter (see [Table 3-230](#) (page 302)). If the file was not previously opened for writing, or the handle is invalid (invalid handle error), or the

`writeSize` exceeds the iPod's capabilities (bad parameter error), then the operation will fail. If the caller attempts to write too much data, the state of the file will be undefined; some or none of the data may have been added to the file.

All data must be written sequentially, but write actions may occur across multiple `WriteiPodFileData` commands. The amount of data transferred must also be within the bounds set by the iPod's capabilities. If any of these criteria are not met, the iPod will return an `iPodACK` command with a failure message.

When a `WriteiPodFileData` command fails, the state of the file is left unknown. The accessory must close the file and then create a new file to write the data.

`WriteiPodFileData` passes the following parameters:

- `offset` is the offset (in bytes) into the file at which to begin writing.
- `handle` is a unique file identifier.
- `data` is the data to be written to the file.

**Table 3-234** `WriteiPodFileData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x07	Command ID: <code>WriteiPodFileData</code>
5	0xNN	offset (bits 31:24). The offset into the file at which to begin writing, in bytes.
6	0xNN	offset (bits 23:16)
7	0xNN	offset (bits 15:8)
8	0xNN	offset (bits 7:0)
9	0xNN	handle (1 byte).
10...N	0xNN	data (variable length). The file data.
(last byte)	0xNN	Checksum

## Command 0x08: `CloseiPodFile`

Direction: Device to iPod

This command closes a file and releases its handle. The handle is invalid for further use after this call, and the iPod may assign the handle later to represent a different file. An `iPodACK` command is sent on success or failure. Parameter `handle` is a unique file identifier.

**Table 3-235** CloseiPodFile packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x08	Command ID: CloseiPodFile
5	0xNN	handle (1 byte).
6	0xNN	Checksum

## Command 0x10: GetiPodFreeSpace

---

Direction: Device to iPod

The accessory asks the iPod to return the amount of free space on its storage system. The iPod replies with RetiPodFreeSpace.

**Table 3-236** GetiPodFreeSpace packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x10	Command ID: GetiPodFreeSpace
5	0xE2	Checksum

## Command 0x11: RetiPodFreeSpace

---

Direction: iPod to Device

The iPod tells the accessory the current amount of free space (in bytes) in its storage system.

**Table 3-237** RetiPodFreeSpace packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0A	Length of packet
3	0x0C	Lingo ID: Storage lingo
4	0x11	Command ID: RetiPodFreeSpace
5	0xNN	freeSpace (bits 63:56). The amount of the iPod's free space.
6	0xNN	freeSpace (bits 55:48)
7	0xNN	freeSpace (bits 47:40)
8	0xNN	freeSpace (bits 39:32)
9	0xNN	freeSpace (bits 31:24)
10	0xNN	freeSpace (bits 23:16)
11	0xNN	freeSpace (bits 15:8)
12	0xNN	freeSpace (bits 7:0)
13	0xNN	Checksum

## Command 0x12: OpeniPodFeatureFile

---

Direction: Device to iPod

The accessory uses the `OpeniPodFeatureFile` command to open a feature file on the iPod. [Table 3-238](#) (page 308) presents the format of the `OpeniPodFeatureFile` command packet.

In response, the iPod returns the `iPodACK` command with the operation status and, if the command succeeded, the feature file handle. If the feature type is not supported or the feature parameters are not valid, an `iPodACK` with bad parameter error status is returned. with an invalid file handle.

**Table 3-238** OpeniPodFeatureFile packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0xNN	Length of Packet
3	0x0C	Lingo ID: Storage



Byte number	Value	Comment
4	0x12	Command ID: OpeniPodFeatureFile
5	0xNN	featureType
6	0xNN	fileOptionsMask (bits 31:24)
7	0xNN	fileOptionsMask (bits 23:16)
8	0xNN	fileOptionsMask (bits 15:8)
9	0xNN	fileOptionsMask (bits 7:0)
10...N	0xNN	fileData (maximum 128 bytes) data to be appended at close. fileOptionsMask bit 0 must be set if this field is nonzero length.
(last byte)	0xNN	Checksum

**Table 3-239** (page 309) lists the current possible `featureType` values for the `OpeniPodFeatureFile` command packet.

**Table 3-239** featureType values

featureType	Description
0x00	Reserved
0x01	Radio tagging (no <code>fileOptionsMask</code> or <code>fileData</code> can be passed)
0x02	Cardio equipment workout
0x03-0xFF	Reserved

**Table 3-240** (page 309) lists the current possible `fileOptionsMask` values for the `OpeniPodFeatureFile` command.

**Table 3-240** fileOptionsMask values

fileOptionsMask	Description
bit 0	<p>1 = On feature file close or device detach, append the <code>fileData</code> binary file data bytes to the file. Note that <code>fileData</code> must be nonzero length if this bit is set.</p> <p>0 = Do not append any bytes to the file. <code>fileData</code> must be zero length if this bit is zero.</p>
bit 1	<p>1 = On file close or device detach, append the XML <code>&lt;ipodInfo&gt;</code> element to the file (includes <code>&lt;openTime&gt;</code>, <code>&lt;closeTime&gt;</code>, <code>&lt;model&gt;</code>, <code>&lt;softwareVersion&gt;</code>, and <code>&lt;serialNumber&gt;</code>). This option is applied before the bit 0 option.</p> <p>0 = Do not write <code>&lt;ipodInfo&gt;</code> element to file.</p> <p>This option must be set when the XML Signature (bit 3) option is set = 1.</p>

fileOptionsMask	Description
bit 2	Reserved (must be 0).
bit 3	1 = On file close or device detach, insert an XML <Signature> element to the file. This option is applied after bit 0 and bit 1 options. 0 = Do not insert an XML <Signature> element. When this option bit is set, the <ipodInfo> option (bit 1) must be set = 1.
bits 4-31	Reserved (must be 0).

## Command 0x80: DeviceACK

Direction: Device to iPod

The accessory must send the `DeviceACK` command to acknowledge the receipt of a Storage lingo command from the iPod. The accessory must respond with bad parameter status whenever a Storage lingo command in the range 0x83 to 0xFF is received from the iPod.

[Table 3-241](#) (page 310) shows the format of the `DeviceACK` command packet.

**Table 3-241** `DeviceACK` packet

Byte number	Value	
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet
3	0x0C	Lingo ID: Storage
4	0x80	Command ID: <code>DeviceACK</code>
5	0xNN	<code>ackStatus</code> of command
6	0xNN	ID of Command being acknowledged
7	0xFF	File Handle for Command (set to 0xFF for not applicable)
8	0xNN	Checksum

[Table 3-242](#) (page 310) presents the possible values for the `ackStatus` byte.

**Table 3-242** `ackStatus` values

ackStatus	Description
0x00	Success

ackStatus	Description
0x01	N/A (reserved)
0x02	Command failed
0x03	Out of resources
0x04	Bad parameter
0x05-0xFF	Reserved

## Command 0x81: GetDeviceCaps

---

Direction: iPod to Device

The `GetDeviceCaps` command may be sent by the iPod to ask the device to return its storage capabilities (if any). [Table 3-243](#) (page 311) presents the format of the `GetDeviceCaps` command packet. The device replies with `RetDeviceCaps`.

**Table 3-243** `GetDeviceCaps` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)
2	0x02	Length of Packet
3	0x0C	Lingo ID: Storage
4	0x81	Command ID: <code>GetDeviceCaps</code>
5	0x71	Checksum

## Command 0x82: RetDeviceCaps

---

Direction: Device to iPod

The accessory must send the `RetDeviceCaps` command in response to a `GetDeviceCaps` command from the iPod to determine the storage capabilities supported by the device. [Table 3-244](#) (page 311) presents the format of the `RetDeviceCaps` command packet.

**Table 3-244** `RetDeviceCaps` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of Packet (SOP)

Byte number	Value	Comment
2	0x18	Length of Packet
3	0x0C	Lingo ID: Storage
4	0x82	Command ID: RetDeviceCaps
5 - 23	0x00	Reserved (must be 0x00)
24	0xFF	Reserved (must be 0xFF)
25	0xNN	Major Storage lingo protocol version supported by device (currently 0x01)
26	0xNN	Minor Storage lingo protocol version supported by device (currently 0x02)
27	0xNN	Checksum

## Lingo 0x0E: Location Lingo

The National Marine Electronics Association (NMEA) has established a standard interface for transferring global positioning system (GPS) information. The NMEA *0183 Interface Standard*, Version 3.01 (released 2002/01), gives complete details about the sentence formats and contents.

The iAP Location lingo provides a mechanism by which NMEA sentences and other types of location information can be sent from accessories to attached iPods or iPhones. Thus the Location lingo lets an accessory that generates raw location data send the data to an iPod that collects, interprets, and displays it. Currently, only the iPhone OS 3.0 supports the Location lingo.

**Note:** Accessories that include GPS receivers must support the Location lingo in addition to any other lingoes they support.

## Location Data Requirements

Accessories that generate NMEA GPS location sentences must support at least the GPGLA sentence type, as defined in the *0183 Interface Standard*. The iPod also accepts the GPRMC sentence type; if the accessory supports it, the accessory must provide it to the iPod when using the Location lingo. The accessory may support additional NMEA sentence types, either standard or proprietary, and future iPod models may also support more sentence types.

For maximum quality of the user experience, accessories that generate NMEA GPS location sentences must enable filtering and pass to the iPod only sentence types that are in the filter list.

## Accessory Power Using the Location Lingo

---

Attached accessories that support the Location lingo are notified of iPod state changes, such as transitions between Power On and Hibernate. The accessory must keep its power consumption below the maximum allowed limits for each iPod state; see ["iPod Power States and Accessory Power"](#) (page 341). Accessory power will be off when the iPod hibernates; when the iPod wakes, the accessory must identify and authenticate itself, using the IDPS process specified in ["Device Signaling and Initialization"](#) (page 48).

Accessories that support the Location lingo may register for intermittent high power during their identification process. If so registered, they may draw up to 100 mA after receiving a `SetDevControl` command specifying GPS radio power on (see [Table 3-258](#) (page 324)). Accessories must reduce their power consumption below 5 mA within 1 second of receiving a `SetDevControl` command specifying GPS radio power off. Regardless of the state of power set by the Location lingo, accessories must comply with iPod state changes as detailed in ["iPod Power States and Accessory Power"](#) (page 341).

## Command Summary

---

Certain prerequisites must be met for an iPhone OS 3.0 accessory to use the Location lingo:

- The accessory must identify itself through the Identify Device Preferences and Settings (IDPS) process, as specified in ["Accessory Identification"](#) (page 445).
- To send and receive Location lingo and IDPS commands, the accessory must be able to generate and interpret transaction IDs, as described in ["Transaction IDs"](#) (page 451), throughout the communication session.
- The accessory must perform Authentication 2.0, as described in ["Authentication"](#) (page 52). Location lingo commands are usable after the accessory enters the background authentication state.

**Note:** With iPods that support wireless iAP over Bluetooth, an accessory need not be physically attached to an iPod to send it location information.

The iPod determines accessory capabilities by sending the Location lingo `GetDevCaps` command. A type parameter in this command specifies the type of capabilities queried. Capabilities type 0x00 requests the types of location services that the accessory supports and types 0x01–0x3F request details about the specific location services. Location capabilities include the following:

- Accessory general capabilities
- NMEA GPS location information (sentence filtering and GPGLL sentence generation are required).
- Location assistance information
- Asynchronous location notification control
- Capabilities specific to each location information type

Location lingo commands are extensible; new features, capabilities, and parameters may be added to future lingo versions. Lingo extensions will be made backward compatible so that existing implementations will continue to work. Accessory implementations must comply with the following rules to be compatible with future lingo versions:

- For all command responses, check for a packet length greater than or equal to the expected length. Do not check for an exact payload length, because this will fail after extensions are made. If the packet length is greater than expected, ignore any additional data.
- When reading lingo bitfield parameters, clear unrecognized bits to 0 before testing for feature bits.
- For other parameters, do range checks on their values and return a `DevACK` command with a bad parameter status (0x04) if a parameter is out of range.

**Note:** iPods and accessories that support the Location lingo must initialize their control states to empty or disabled when the iPod-accessory connection is first established. Initializing the control state is effectively the same as receiving `SetDevControl` commands for all `locType` values, with `ctlData` set to 0x0 for each. This will ensure that for all `locType` values, notifications are disabled by default and accessories are in a low power state. One or more control states may be changed later, depending on the iPod's or accessory's capabilities and usage requirements.

Table 3-245 (page 314) lists the Location lingo commands.

**Table 3-245** Location lingo commands

ID	Name	Direction	Parameters
0x00	DevACK	Dev to iPod	{transID;2, cmdStatus;1, cmdIDOrig;1}
0x01	GetDevCaps	iPod to Dev	{transID;2, locType;1}
0x02	RetDevCaps	Dev to iPod	{transID;2, locType;1, capsData;<var>}
0x03	GetDevControl	iPod to Dev	{transID;2, locType;1}
0x04	RetDevControl	Dev to iPod	{transID;2, locType;1, ctlData;<var>}
0x05	SetDevControl	iPod to Dev	{transID;2, locType;1, ctlData;<var>}
0x06	GetDevData	iPod to Dev	{transID;2, locType;1, dataType;1}
0x07	RetDevData	Dev to iPod	{transID;2, locType;1, dataType;1, sectCur;2, sectMax;2, locData;<var>}
0x08	SetDevData	iPod to Dev	{transID;2, locType;1, dataType;1, sectCur;2, sectMax;2, locData;<var>}
0x09	AsyncDevData	Dev to iPod	{transID;2, locType;1, dataType;1, sectCur;2, sectMax;2, locData;<var>}
0x0A–0x7F	Reserved		
0x80	iPodACK	iPod to Dev	{transID;2, cmdStatus;1, cmdIDOrig;1}
0x81–0xFF	Reserved		

## A Typical Location Data Session

[Table 3-246](#) (page 315) shows a typical exchange of commands by which an iPod may obtain location data from an accessory. The first time such session could take place would be after the iPod had acknowledged successful completion of the accessory identification process by sending `IDPSStatus` (see [Accessory Identification](#) (page 445)) and had begun the process of authenticating the accessory, as specified in ["Authentication"](#) (page 52).

**Table 3-246** Sample command interchange

Step	Command	locType	dataType	Direction	Comments
1	GetDevCaps	0x00		iPod to Dev	The iPod queries the accessory's system capabilities.
2	RetDevCaps	0x00		Dev to iPod	The accessory returns its system capabilities.
3	GetDevCaps	0x01		iPod to Dev	The iPod queries the accessory's NMEA GPS location capabilities.
4	RetDevCaps	0x01		Dev to iPod	The accessory returns its NMEA GPS location capabilities.
5	GetDevCaps	0x02		iPod to Dev	The iPod queries the accessory's location assistance capabilities.
6	RetDevCaps	0x02		Dev to iPod	The accessory returns its location assistance capabilities.
7	GetDevControl	0x00		iPod to Dev	The iPod queries the accessory's system control state.
8	RetDevControl	0x00		Dev to iPod	The accessory returns its system control state.
9	GetDevControl	0x01		iPod to Dev	The iPod queries the accessory's NMEA GPS location control state.
10	RetDevControl	0x01		Dev to iPod	The accessory returns its NMEA GPS location control state.
11	SetDevControl	0x00		iPod to Dev	The iPod sets the accessory's system control state for <code>locType = 0x00</code> in accordance with the capabilities and controls sent to it by previous <code>RetDevCaps</code> and <code>RetDevControl</code> commands.
12	DevACK			Dev to iPod	The accessory acknowledges receipt of <code>SetDevControl</code> .
13	SetDevData	0x01	0x00	iPod to Dev	The iPod sets the NMEA sentence filter list string.

Step	Command	locType	dataType	Direction	Comments
14	DevACK			Dev to iPod	The accessory acknowledges receipt of SetDevData.
15	SetDevControl	0x01		iPod to Dev	The iPod sets the accessory's system control state for locType = 0x01 in accordance with the capabilities and controls sent to it by previous RetDevCaps and RetDevControl commands.
16	DevACK			Dev to iPod	The accessory acknowledges receipt of SetDevControl.
17	GetDevData	0x02	0x03	iPod to Dev	The iPod requests the satellite ephemeris maximum refresh interval.
18	RetDevData	0x02	0x03	Dev to iPod	The accessory returns the satellite ephemeris maximum refresh interval.
19	GetDevData	0x02	0x04	iPod to Dev	The iPod requests the satellite ephemeris recommended refresh interval.
20	RetDevData	0x02	0x04	Dev to iPod	The accessory returns the satellite ephemeris recommended refresh interval.
21	AsyncDevData	0x02	0x05	Dev to iPod	The accessory sends a request for current GPS time of the system.
22	iPodACK			iPod to Dev	The iPod acknowledges receipt of the GPS time request.
23	AsyncDevData	0x02	0x02	Dev to iPod	The accessory sends an ephemeris URL to request an update to its ephemeris data.
24	iPodACK			iPod to Dev	The iPod acknowledges receipt of the satellite ephemeris data URL string.
25	SetDevData	0x02	0x05	iPod to Dev	The iPod sets the current GPS time of the accessory.
26	DevACK			Dev to iPod	The accessory acknowledges receipt of SetDevData.
27	SetDevData	0x02	0x00	iPod to Dev	The iPod sets the current location data on the accessory for use in looking up the satellite position within the ephemeris.
28	DevACK			Dev to iPod	The accessory acknowledges receipt of SetDevData.



Step	Command	locType	dataType	Direction	Comments
29	SetDevData	0x02	0x01	iPod to Dev	The iPod sets the accessory's ephemeris with data downloaded from the URL provided in Step 23.
30	DevACK			Dev to iPod	The accessory acknowledges receipt of SetDevData.
If multisection data, repeat Steps 29–30 as needed. See <a href="#">"Multisection Data Transfers"</a> (page 459).					
31	AsyncDevData	0x01	0x80	Dev to iPod	The accessory begins sending location data to the iPod.
32	iPodACK			iPod to Dev	The iPod acknowledges receipt of AsyncDevData.
If multisection data, repeat Steps 31–32 as needed. See <a href="#">"Multisection Data Transfers"</a> (page 459).					

## Command 0x00: DevACK

Direction: Accessory to iPod

This command is sent by the accessory in response to a command sent from the iPod. It has two forms:

- The Section form is sent if the command from the iPod is part of a multisection data transfer. This form of the command is discussed in ["Multisection DevACK Command"](#) (page 460).
- The Default form, shown in [Table 3-247](#) (page 317), is sent under these conditions:
  - ❑ When a bad parameter is received, an unsupported or invalid command is received, or a command that does not return any data has completed.
  - ❑ To indicate completion of a multisection data transfer, as described in ["Multisection Data Transfers"](#) (page 459).

**Table 3-247** Default DevACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x00	Command ID: DevACK
5	0xNN	transID [bits 15:8]: Transaction ID of command being acknowledged; see <a href="#">"Transaction IDs"</a> (page 451)

Byte number	Value	Comment
6	0xNN	transID [bits 7:0]
7	0xNN	ackStatus : Status return; possible values are listed in <a href="#">Table 3-248</a> (page 318)
8	0xNN	cmdIDorig : Original command ID for which this response is being sent.
9	0xNN	Checksum

**Table 3-248** Valid Location lingo ackStatus values

Value	Meaning
0x00	Command OK
0x02	Command failed (valid command but did not succeed)
0x03	Out of resources (iPod internal allocation failed)
0x04	Bad parameter (invalid command or input parameters)
0x06	Reserved
0x0F	Command timeout
0x13	Section of multisection data transfer received successfully (see " <a href="#">Multisection Data Transfers</a> " (page 459))

## Command 0x01: GetDevCaps

Direction: iPod to Accessory

This command is sent by the iPod to get the accessory's capabilities. In response, the accessory sends a `RetDevCaps` command with the same `transactionID`, the capability type, and the requested capability data. If a specified `locType` is not supported, a `DevACK` must be returned with the bad parameter (0x04) status. Accessory support for capability type 0x00 (system caps) is mandatory; other capability types are optional. The iPod requests the system caps from the accessory to determine its lingo version, system capabilities, and the other location types supported by the accessory. The accessory must send a corresponding `RetDevCaps` within 500 ms in response to the `GetDevCaps` command from the iPod.

**Table 3-249** GetDevCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x0E	Lingo ID: Location lingo

Byte number	Value	Comment
4	0x01	Command ID: GetDevCaps
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 451)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The capability type; see Table 3-250 (page 319).
8	0xNN	Checksum

**Table 3-250** Values for locType

Value	Meaning
0x00	System capabilities; see Table 3-252 (page 320).
0x01	NMEA GPS location capabilities; see Table 3-253 (page 321).
0x02	Location assistance capabilities; see Table 3-254 (page 321).
0x03–0x3F	Reserved
0x40–0xFF	Invalid

## Command 0x02: RetDevCaps

Direction: Accessory to iPod

The accessory sends this command in response to a `GetDevCaps` command from the iPod, returning the transaction ID and the requested capabilities type. It informs the iPod of its capabilities in `capsData`. If an accessory does not support a particular capability type, it must return a `DevACK` command with a bad parameter (0x04) status.

The `capsData` fields may be extended in the future as new features are added to the Location lingo. iPods receiving this command check for a minimum packet payload length (rather than an exact payload length) and parse only the supported fields, ignoring any additional data that may be appended.

**Table 3-251** RetDevCaps packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x02	Command ID: RetDevCaps

Byte number	Value	Comment
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 451)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The capability type; see Table 3-250 (page 319).
8–NN	0xNN	capsData: Capabilities data; see Table 3-252 (page 320), Table 3-253 (page 321), and Table 3-254 (page 321). The contents and length of this parameter depend on the value of locType.
NN+1	0xNN	Checksum

**Table 3-252** System capabilities values (locType = 0x00)

Name	Size in bytes	Bits	Meaning
devVerMajor	1	Bits 7:0	Location lingo major version supported by the accessory
devVerMinor	1	Bits 7:0	Location lingo minor version supported by the accessory
sysCapsMask	8	System capabilities (big-endian):	
		Bit 00	1 = Power management control support; iPod-powered accessories must set this bit to allow power control by the iPod.
		Bit 01	Reserved; set to 0.
		Bit 02	1 = Asynchronous location notification support
		Bits 63:03	Reserved; set to 0.
locCapsMask	8	Location type support (big-endian):	
		Bit 00	1 = System support; must be set to 1.
		Bit 01	1 = NMEA GPS location support (see <b>Note</b> below)
		Bit 02	1 = Location assistance support; either this bit or bit 01 must be set
		Bits 63:03	Reserved; set to 0.

**Note:** Accessories that generate NMEA GPS location sentences must support at least the GPGLA sentence type. The iPod also accepts the GPRMC sentence type; if the accessory supports it, the accessory must provide it to the iPod when using the Location lingo. The accessory may support additional NMEA sentence types, either standard or proprietary. Sentences must be generated at intervals compliant with the NMEA 0183 *Interface Standard*, unless overridden by system controls or NMEA sentence filtering.

**Table 3-253** NMEA GPS location capabilities values (locType = 0x01)

Name	Size in bytes	Bits	Meaning
nmeaGpsCaps	8	Bit 00 (big-endian)	Must be set to 1 to indicate NMEA GPS sentence filtering supported; the accessory must support a minimum filter list string length of 128 bytes.
		Bits 63:01	Reserved; set to 0.

**Table 3-254** Location assistance capabilities values (locType = 0x02)

Name	Size in bytes	Bits	Meaning
locAsstData	8	Location assistance data types supported; the accessory's most recent <code>RetDevCaps</code> command must set the <code>sysCapsMask</code> capability bit 01 if one or more of the following bits are set (see <a href="#">Table 3-252</a> (page 320)):	
		Bit 00 (big-endian)	Reserved; set to 0.
		Bit 01	Satellite ephemeris data required, for faster location fix. This bit must be set if bit 02 is set.
		Bit 02	Satellite ephemeris data URL string (RFC 1738 compliant). This bit must be set if bit 01 is set. The accessory must supply a Web URL string for the iPod to use to download ephemeris data and must accept that data when it is set by the iPod (see <a href="#">Command 0x06: GetDevData</a> (page 325) and <a href="#">Command 0x08: SetDevData</a> (page 328)).
		Bit 03	Satellite position ephemeris data maximum refresh interval. If the accessory's long-term ephemeris degrades over time, this indicates the maximum amount of time during which downloaded data will be valid.
		Bit 04	Satellite position ephemeris data recommended refresh interval. If the accessory's long term ephemeris degrades over time, this indicates the recommended amount of time before a new download is needed.
		Bits 63:05	Reserved; set to 0.

## Command 0x03: GetDevControl

Direction: iPod to Accessory

This command is sent by the iPod to get the accessory's control state. In response, the accessory must send a `RetDevControl` command within 500 ms, passing the same transaction ID and location type plus the accessory's control state information. A control type is supported only if the accessory's most recent `RetDevCaps` command set the `locCapsMask` capability bit. If the accessory receives an unsupported control type, it must return a `DevACK` command with a bad parameter (0x04) status.

**Table 3-255** GetDevControl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x05	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x03	Command ID: <code>GetDevControl</code>
5	<i>NN</i>	<code>transID</code> [bits 15:8]: Transaction ID; see "Transaction IDs" (page 451)
6	<i>NN</i>	<code>transID</code> [bits 7:0]
7	<i>NN</i>	<code>locType</code> : The location control type; see Table 3-250 (page 319).
8	<i>NN</i>	Checksum

## Command 0x04: RetDevControl

Direction: Accessory to iPod

This command is sent by the accessory in response to a `GetDevControl` command received from the iPod. The transaction ID and control type received from `GetDevControl` must be returned with this command, along with the accessory's current control state of the specified control type. System control support is required for every accessory.

**Note:** The `RetDevControl` `ctlData` fields may be extended in the future as new features are added. iPods receiving this command check for a minimum packet payload length (rather than an exact payload length) and parse only the supported fields, ignoring any unrecognized additional data that may be appended.

**Table 3-256** RetDevControl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet (SOP)
2	0x0D	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x04	Command ID: RetDevControl
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 451)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The location control type; see Table 3-250 (page 319).
8–15	0xNNNNNNNNNNNNNNNN	ctlData: System control data, depending on the value of locType; see Table 3-258 (page 324).
16	0xNN	Checksum

## Command 0x05: SetDevControl

Direction: iPod to Accessory

This command is sent by the iPod to set the accessory's control state. In response, the accessory must send a DevACK command within 500 ms, passing the same transaction ID and the status of the operation. Control types are valid only if the accessory's most recent RetDevCaps command set the associated RetDevCaps locCapsMask location type bits. If the iPod tries to set reserved or unsupported control types or data, the accessory must return a DevACK command with the bad parameter (0x04) status.

Accessories that support the Location lingo may register for intermittent high power during their identification process. If so registered, they may draw up to 100 mA after receiving this command with a ctlData value specifying GPS radio power on; see Table 3-258 (page 324). Accessories must reduce their power consumption below 5 mA within 1 second of receiving this command with a ctlData value specifying GPS radio power off. Regardless of the state of power set by the Location lingo, accessories must comply with iPod state changes as detailed in "iPod Power States and Accessory Power" (page 341).

**Note:** The SetDevControl ctlData fields may be extended in the future as new features are added. iPods implementing the Location lingo check for a minimum packet payload length (rather than an exact payload length) and parse only the supported fields, ignoring any unrecognized additional data that may be appended.

**Table 3-257** SetDevControl packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)

Byte number	Value	Comment
2	0x0D	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x05	Command ID: SetDevControl
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 451)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The location control type; see Table 3-250 (page 319).
8–15	0xNNNNNNNNNNNNNNNN	ctlData: The accessory controls to be set; see Table 3-258 (page 324).
16	0xNN	Checksum

**Table 3-258** ctlData values

locType	Name	Bits	Meaning
0x00	sysCtl	System control data:	
		Bits 01:00	Accessory GPS radio power and notify control state:
			00: Power off; the accessory must turn its GPS radio power off when this control state is set. If the accessory is using iPod power, it must reduce its current consumption to 5 mA or less within 1 second of receiving SetDevControl.
			01–10: Reserved
			11: Power on; the accessory must turn its GPS radio power on when this control state is set. If the accessory is using iPod power, it may draw up to 100 mA after receiving SetDevControl.
0x01	nmeaGpsCtrl	Bit 02	1 = Asynchronous location notifications enabled, 0 = notifications disabled.
		Bits 63:03	Reserved; set to 0.
0x01	nmeaGpsCtrl	Bit 00	When set to 1, NMEA GPS sentence filtering is enabled. When set to 0, NMEA GPS filtering is disabled; if asynchronous location notifications are enabled, all sentences must be passed to the iPod.
		Bits 63:01	Reserved; set to 0.
0x02–0x3F	Reserved		



locType	Name	Bits	Meaning
0x40–0xFF	Invalid		

## Command 0x06: GetDevData

Direction: iPod to Accessory

This command is sent by the iPod to get location data of a specific type from the accessory. In response, the accessory sends a `RetDevData` command with the same transaction ID, location type, and the requested location data information. The accessory must send the corresponding `RetDevData` command within 500 ms after receiving a `GetDevData` command from the iPod. In the case of multi-section responses, accessories must send each subsequent section within 500 ms of sending the previous section.

**Table 3-259** GetDevData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x06	Command ID: <code>GetDevData</code>
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 451)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The capability type; see Table 3-250 (page 319).
8	0xNN	dataType: Location data type; See Table 3-260 (page 325).
9	0xNN	Checksum

**Table 3-260** GetDevData dataType values

locType	Name	Data type	Meaning
0x00–0x01	Reserved		
0x02	locAsstData	Location assistance data; this data is available only if the accessory's most recent <code>RetDevCaps</code> command set the <code>locCapsMask</code> capability bit 01. The specific location data types are valid only if the associated <code>RetDevCaps</code> <code>locAssistance</code> bits were also set.	
		0x00–0x02	Reserved

locType	Name	Data type	Meaning
		0x03	Satellite ephemeris data maximum required refresh interval. If the accessory's long-term ephemeris degrades over time, this indicates the maximum amount of time during which downloaded data will be valid.
		0x04	Satellite ephemeris data recommended refresh interval. If the accessory's long-term ephemeris degrades over time, this indicates the recommended amount of time before a new download is needed.
		0x05–0xFF	Reserved
0x03–0x3F	Reserved		
0x40–0xFF	Invalid		

## Command 0x07: RetDevData

Direction: Accessory to iPod

This command is sent by the accessory in response to a `GetDevData` command received from the iPod, returning its transaction ID, `locType`, and `dataType`, indicating the data type requested. The `locData` field is variable in length, based on the location data types listed in [Table 3-260](#) (page 325). The iPod acknowledges `RetDevData` with an `iPodACK` command.

The iPod has a maximum input data payload size of 500 bytes including the transaction ID. If the `locData` value sent by the accessory exceeds this size, it must be split into multiple sections as described in [Multisection Data Transfers](#) (page 459).

**Table 3-261** RetDevData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x07	Command ID: RetDevData
5	0xNN	transID [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The capability type; see <a href="#">Table 3-262</a> (page 327).

Byte number	Value	Comment
8	0xNN	dataType: Location data type; see <a href="#">Table 3-262</a> (page 327).
9	0xNN	sectCur [bits 15:8]: Current payload section index (0x0000 = first or only section)
10	0xNN	sectCur [bits 7:0]
11	0xNN	sectMax [bits 15:8]: Maximum payload section index (0x0000 = only section, 0xNNNN = maximum or last section)
12	0xNN	sectMax [bits 7:0]
13	0xNN	totSize [bits 31:24]: Total size of locData in bytes. If multiple RetDevData packets are sent, the totSize field is included only in the first packet (sectCur = 0x0000).
14	0xNN	totSize [bits 23:16]
15	0xNN	totSize [bits 15:8]
16	0xNN	totSize [bits 7:0]
13–NN or 17–NN	<var>	locData: See <a href="#">Table 3-262</a> (page 327).
(last byte)	0xNN	Checksum

**Table 3-262** RetDevData locData values

locType	Name	Meaning
0x00–0x01	Reserved	
0x02	locAsstData	Location assistance data: see <a href="#">Table 3-263</a> (page 327). This data type is valid only if the accessory's most recent RetDevCaps command set the sysCapsMask capability bit 01.
0x03–0x3F	Reserved	
0x40–0xFF	Invalid	

**Table 3-263** locAsstData values (locType = 0x02) for RetDevData

dataType	Bytes	Field	Bytes	Subfield	Bytes	Value
0x00–0x02	Reserved.					
0x03	4	Satellite position ephemeris data maximum refresh interval, in milliseconds. Ephemeris data must be refreshed at least as often as this interval. This field must be sent in a single packet (sectCur = 0x0000 and sectMax = 0x0000).				

dataType	Bytes	Field	Bytes	Subfield	Bytes	Value
0x04	4	Satellite position ephemeris data recommended refresh interval, in milliseconds. Ephemeris data should be refreshed at least as often as this interval. This field must be sent in a single packet (sectCur = 0x0000 and sectMax = 0x0000). Its value must be less than or equal to the maximum refresh interval (dataType 0x03).				
0x05–0x3F	Reserved.					
0x40–0xFF	Invalid.					

## Command 0x08: SetDevData

Direction: iPod to Accessory

This command is sent by the iPod to set location data on the accessory. In response, the accessory must return a `DevACK` command within 500 ms with the received transaction ID and the status of this operation. This time limit applies to both single and multisection responses. The `dataType` field represents the data type to be set; only certain data types, listed in [Table 3-265](#) (page 329), can be set. The `locData` field is variable in length, based on the location data types listed in [Table 3-267](#) (page 330).

The accessory should return its maximum payload size in its IDPS `accInfo` token; see [Table 2-76](#) (page 114). If the accessory does not specify a maximum payload size, the iPod assumes a default size of 1024 bytes. If the `locData` value to be sent by the iPod exceeds this size, it will be split into multiple sections as described in [Multisection Data Transfers](#) (page 459).

**Table 3-264** SetDevData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x08	Command ID: SetDevData
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451)
6	0xNN	<code>transID</code> [bits 7:0]
7	0xNN	<code>locType</code> : The capability type; see <a href="#">Table 3-265</a> (page 329).
8	0xNN	<code>dataType</code> : the data types that can be set are listed in <a href="#">Table 3-265</a> (page 329).
9	0xNN	<code>sectCur</code> [bits 15:8]: Current payload section index (0x0000 = first or only section)
10	0xNN	<code>sectCur</code> [bits 7:0]

Byte number	Value	Comment
11	0xNN	sectMax [bits 15:8]: Maximum payload section index (0x0000 = only section, 0xNNNN = maximum or last section)
12	0xNN	sectMax [bits 7:0]
13	0xNN	totSize [bits 31:24]: Total size of locData in bytes. If multiple SetDevData packets are sent, the totSize field is included only in the first packet (sectCur = 0x0000).
14	0xNN	totSize [bits 23:16]
15	0xNN	totSize [bits 15:8]
16	0xNN	totSize [bits 7:0]
13–NN or 17–NN	<var>	locData: see <a href="#">Table 3-267</a> (page 330).
(last byte)	0xNN	Checksum

**Table 3-265** Data types settable by SetDevData

locType	dataType	Description
0x01	0x00	NMEA sentence filter list string; see <a href="#">Table 3-266</a> (page 329).
0x02	0x00	Current point location data; see <a href="#">Table 3-267</a> (page 330).
0x02	0x01	Satellite ephemeris data; see <a href="#">Table 3-267</a> (page 330).
0x02	0x05	Current GPS time on accessory; see <a href="#">Table 3-267</a> (page 330).

**Table 3-266** nmeaGpsLocData values (locType = 0x01)

dataType	Value
0x00	<p>NMEA sentence filter list string, a comma-delimited set of NMEA standard and/or proprietary uppercase acronyms for sentence types. The accessory must support a filter string of at least 128 bytes. Depending on the string length, the iPod may split this command into multiple sections. Only the last section will include the string null terminator.</p> <p>An example sentence list string could be GPGGGA, GPRMC, specifying the two sentence types to be enabled. The accessory must not send sentence types not in the filter list. A string consisting of only the null terminator disables the sending of all NMEA sentences by the accessory.</p>
0x01–0xFF	Reserved

**Table 3-267** locAsstData values (locType = 0x02) for SetDevData

dataType	Bytes	Field	Bytes	Subfield	Bytes	Value
0x00	16	Current point location data; must be sent as a single packet (sectCur = 0x0000 and sectMax = 0x0000). See <b>Note</b> , below.				
		locWeek	2	GPS Week number. The 13 least significant bits comprise a modulo-8192 representation of the current GPS Week number, as defined by IS-GPS-200 PIRN-002, Section 30.3.3.1.1.1. The GPS Week Number count began at UTC midnight Saturday-Sunday January 5-6, 1980. Since then the count has been broadcast as part of the GPS message and has incremented by 1 each UTC Saturday-to-Sunday transition. The GPS Week number will roll over in the year 2137. The week containing Tuesday, April 28, 2009 is GPS Week 1,529.		
		locTime	4	GPS Time of Week. The 30 least significant bits represent the time when the location point was determined. This is measured as milliseconds in the week from the last UTC Saturday-to-Sunday transition, a number between 0 and 604,800,000.		
		locPoint	8	Current location point in millionths of a degree latitude and longitude		
				latDegArc	4	Signed 32-bit latitude: 0x00000000 = ±0° (Equator), 0x055D4A80 = +90° (maximum north), 0xFAA2B580 = −90° (maximum south).
				lonDegArc	4	Signed 32-bit longitude: 0x00000000 = ±0° (Greenwich meridian), 0x0ABA9500 = +180.000000° (maximum east), 0xF5456B01 = −179.999999° (maximum west).
		locAccuracy	2	Location accuracy radius		
				locRadius	2	Point location accuracy radius in meters; a smaller radius number means a more accurate point location
0x01	<var>	ephData	<var>	Satellite ephemeris data. This data does not have a standard format and is not parsed by the Location lingo.		
0x02–0x04		Reserved.				
0x05	6	The iPod’s current GPS time.				

dataType	Bytes	Field	Bytes	Subfield	Bytes	Value
		locWeek	2			GPS Week number. The 13 least significant bits comprise a modulo-8192 representation of the iPod's current GPS Week number, as defined by IS-GPS-200 PIRN-002, Section 30.3.3.1.1.1. The GPS Week Number count began at UTC midnight Saturday-Sunday January 5-6, 1980. Since then the count has been broadcast as part of the GPS message and has incremented by 1 each UTC Saturday-to-Sunday transition. The GPS Week number will roll over in the year 2137. The week containing Tuesday, April 28, 2009 is GPS Week 1,529.
		locTime	4			GPS Time of Week. The 30 least significant bits represent the iPod's current GPS time. This is measured as milliseconds in the week from the last UTC Saturday-to-Sunday transition, a number between 0 and 604,800,000.
0x06–0x3F	Reserved.					
0x40–0xFF	Invalid.					

**Note:** The iPod pushes system GPS time and point location data to the accessory, to be used in conjunction with ephemeris data. The GPS time of the iPod's point location data may be significantly behind system time, because it represents the last location lock that the iPod was able to achieve. When calculating its reference into ephemeris data, the accessory should consider both the system time and the point location time to validate the iPod's data.

The iPod pushes system time to the accessory when launching location services or in response to accessory inquiries. Once ephemeris data has been requested the iPod continues to push point location data asynchronously, as the data accuracy changes. It also delivers updated ephemeris data periodically, as available, after the accessory makes its initial request.

## Command 0x09: AsyncDevData

Direction: Accessory to iPod

This command is sent by the accessory to notify the iPod that location data is available. It is valid only if the accessory supports asynchronous notifications (see [Table 3-258](#) (page 324)) and the notification category has been enabled by a `SetDevControl` command. The `dataType` field represents the data type being sent; the currently valid types are listed in [Table 3-269](#) (page 333). The `locData` field is variable in length, depending on the data being sent.

No iAP command may exceed a value of 500 bytes in its length-of-packet-payload field. If the `locData` value sent by the accessory exceeds this size, it must be split into multiple sections with the same transaction ID, as described in [Multisection Data Transfers](#) (page 459). The iPod acknowledges each `AsyncDevData` packet with an `iPodACK` command.

The accessory may send an `AsyncDevData` command with a `dataType` of 0x02 or 0x05 any time Accessory Power is high (pin 13 of the 30-pin connector), regardless of the iPod's control state as set by `SetDevControl`. The iPod responds immediately with an `iPodACK` command, but the data requested will not be sent until iPod location services are actually engaged by an application. The accessory must respond to other iAP

commands while waiting for a response. The URL must not be sent to the iPod more often than the maximum ephemeral refresh interval established by the accessory via `RetDevData`, and in any case not more often than once every 5 minutes.

**Table 3-268** AsyncDevData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0xNN	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x09	Command ID: AsyncDevData
5	0xNN	transID [bits 15:8]: Transaction ID; see "Transaction IDs" (page 451)
6	0xNN	transID [bits 7:0]
7	0xNN	locType: The capability type; see Table 3-269 (page 333).
8	0xNN	dataType: Location data type; see Table 3-269 (page 333).
9	0xNN	sectCur [bits 15:8]: Current payload section index (0x0000 = first or only section)
10	0xNN	sectCur [bits 7:0]
11	0xNN	sectMax [bits 15:8]: Maximum payload section index (0x0000 = only section, 0xNNNN = maximum or last section)
12	0xNN	sectMax [bits 7:0]
13	0xNN	totSize [bits 31:24]: Total size of locData in bytes. If multiple AsyncDevData packets are sent, the totSize field is included only in the first packet (sectCur = 0x0000). See Multisection Data Transfers (page 459).
14	0xNN	totSize [bits 23:16]
15	0xNN	totSize [bits 15:8]
16	0xNN	totSize [bits 7:0]
13–NN or 17–NN	<var>	locData: Data sent by AsyncDevData; see Table 3-269 (page 333).
(last byte)	0xNN	Checksum



**Table 3-269** locData values that can be sent by AsyncDevData

locType	dataType	locData value
0x01	0x80	String of NMEA 0183 compliant sentences. Each sentence must be sent as a full NMEA 0183 compliant sentence including the \$GP, \$P, or !P prefix, the *NN checksum, and the <CR><LF> suffix.
0x02	0x02	Null-terminated URL string (RFC 1738 compliant) to a web site for satellite ephemeris data (such as <a href="http://www.yourcompany.com/long-term-ephemeris">http://www.yourcompany.com/long-term-ephemeris</a> ). This must be a full URL, including <a href="http://">http://</a> . Only http and https transfer protocols are currently supported.
0x02	0x05	Request for current system time in GPS time (GPS Week number and milliseconds offset into the week; see <a href="#">Table 3-267</a> (page 330)).

## Command 0x80: iPodACK

Direction: iPod to Accessory

This command is sent by the iPod in response to a command sent from the accessory. It has two forms:

- The Default form, shown in [Table 3-270](#) (page 333), is sent under these conditions:
  - ❑ When a bad parameter is received, an unsupported or invalid command is received, or a command that does not return any data has completed.
  - ❑ To indicate completion of a multisection data transfer, as described in ["Multisection Data Transfers"](#) (page 459).
- The Section form is sent if the command from the accessory is part of a multisection data transfer. This form of the command is discussed in ["Multisection iPodACK Command"](#) (page 460).

**Table 3-270** Default iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x80	Command ID: iPodACK
5	0xNN	transID [bits 15:8]: Transaction ID of command being acknowledged; see <a href="#">"Transaction IDs"</a> (page 451)
6	0xNN	transID [bits 7:0]
7	0xNN	ackStatus : Status return; possible values are listed in <a href="#">Table 3-248</a> (page 318)

Byte number	Value	Comment
8	0xNN	cmdIDOrig: Original command ID for which this response is being sent.
9	0xNN	Checksum

## Sample Identification Sequences

This section provides the following commented listings of sample command sequences in which an accessory identifies itself to an iPod or iPhone:

- [Table 3-271](#) (page 334) lists sample commands for an accessory that supports the General and Extended Interface lingoes. For Extended Interface command details, see ["The Extended Interface Protocol"](#) (page 341).
- [Table 3-272](#) (page 335) lists sample commands for an accessory that supports the General, Simple Remote, and Display Remote lingoes.
- [Table 3-273](#) (page 337) lists sample commands for an accessory that supports the General, Simple Remote, Display Remote, and Digital Audio lingoes.

**Table 3-271** Identification of lingoes 0x00+0x04

Step	Accessory command	iPod command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'

If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,

- If the accessory receives no ACK command within 1 second, it may retry Step 1 at 1-second intervals as long as iPod Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.
- If the accessory receives an ACK status of 0x04 (Bad Parameter), the iPod does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See ["Cancelling a Current Authentication Process With IdentifyDeviceLingoes"](#) (page 83). A sample command sequence is listed in [Table F-24](#) (page 535).
- If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.

Step	Accessory command	iPod command	Comment
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/4   options 0x00000002   device ID 0x00000200); AccCapsToken = 0x00000000000000205; AccInfoToken = Acc name (Apple); AccInfoToken = Acc FW version (v9.8.7); AccInfoToken = Acc HW version (v1.2.3); AccInfoToken = Acc manufacturer (Apple Inc.); AccInfoToken = Acc model number (ModelNumber-XYZ); SDKProtocolToken = 1 (com.Apple.ProtocolMain); SDKProtocolToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; SDKProtocolToken = (1) accepted; SDKProtocolToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication-Info	no params

**Table 3-272** Identification of lingoes 0x00+0x02+0x03

Step	Accessory command	iPod command	Comment
1	StartIDPS		no params

Step	Accessory command	iPod command	Comment
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> <li>■ If the accessory receives no ACK command within 1 second, it may retry Step 1 at 1-second intervals as long as iPod Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.</li> <li>■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the iPod does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83). A sample command sequence is listed in <a href="#">Table F-25</a> (page 539).</li> <li>■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.</li> </ul>			
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/2/3   options 0x00000002   device ID 0x00000200); AccCapsToken = 0x0000000000000205; AccInfoToken = Acc name (Apple); AccInfoToken = Acc FW version (v9.8.7); AccInfoToken = Acc HW version (v1.2.3); AccInfoToken = Acc manufacturer (Apple Inc.); AccInfoToken = Acc model number (ModelNumber-XYZ); SDKProtocolToken = 1 (com.Apple.ProtocolMain); SDKProtocolToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')

Step	Accessory command	iPod command	Comment
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; SDKProtocolToken = (1) accepted; SDKProtocolToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication-Info	no params

**Table 3-273** Identification of lingoes 0x00+0x02+0x03+0x0A

Step	Accessory command	iPod command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'

If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,

- If the accessory receives no ACK command within 1 second, it may retry Step 1 at 1-second intervals as long as iPod Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.
- If the accessory receives an ACK status of 0x04 (Bad Parameter), the iPod does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See ["Cancelling a Current Authentication Process With IdentifyDeviceLingoes"](#) (page 83). A sample command sequence is listed in [Table F-26](#) (page 542).
- If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.

Step	Accessory command	iPod command	Comment
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/2/3/10   options 0x00000002   device ID 0x00000200); AccCapsToken = 0x00000000000000215; AccInfoToken = Acc name (Apple); AccInfoToken = Acc FW version (v9.8.7); AccInfoToken = Acc HW version (v1.2.3); AccInfoToken = Acc manufacturer (Apple Inc.); AccInfoToken = Acc model number (ModelNumber-XYZ); SDKProtocolToken = 1 (com.Apple.ProtocolMain); SDKProtocolToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; SDKProtocolToken = (1) accepted; SDKProtocolToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication-Info	no params
8	RetDevAuthentication-Info		returns authentication version and X.509 certificate data

Step	Accessory command	iPod command	Comment
9		AckDevAuthentication-Info	acknowledging 'auth info supported'
10		GetAccSampleRateCaps	no params
11		GetDevAuthentication-Signature	sends authentication challenge
12	RetAccSampleRateCaps		returning sample rates '8000   11025   12000   16000   22050   24000   32000   44100   48000'
13	RetDevAuthentication-Signature		returns digital signature in response to authentication challenge
14		NewiPodTrackInfo	sample rate 44100; Sound Check value 0; track volume adjustment 0
15	AccAck		acknowledging 'NewiPodTrackInfo'
16		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
17		NewiPodTrackInfo	sample rate 44100; Sound Check value 0; track volume adjustment 0
18	AccAck		acknowledging 'NewiPodTrackInfo'
19		NewiPodTrackInfo	sample rate 44100; Sound Check value 0; track volume adjustment 0
20	AccAck		acknowledging 'NewiPodTrackInfo'





# The Extended Interface Protocol

---

The iPod Extended Interface protocol allows the user interface of the iPod to be translated to other environments; for example, a vehicle entertainment system such as a dashboard radio, large-screen display, or steering wheel mounted remote control. This document specifies command packets that allow access to the iPod database and device management over serial or USB connections. This interface has the following advantages:

- It allows file-system and database-format independence for the interfacing body.
- It allows the interfacing body to remain ignorant of the encoders and decoders used to process digital audio information.

The minimum iPod System Software versions supported by this specification are:

- Version 2.0 of the third generation (3G) iPod.
- Version 3.0 of the fourth generation (4G) iPod.
- Version 1.0 of the iPod mini, 4G iPod (color display), iPod nano, fifth-generation iPod (5G), second-generation iPod nano, iPod classic, and iPod 3G nano.
- Version 1.1 of the iPhone and the iPod touch.
- Version 2.0.0 of the iPhone 3G.

**Note:** 32- and 16-bit quantities are sent in big-endian form, as the bit breakdowns in this specification show.

## Major iPod Operating Modes

For the purpose of this document, the iPod can be considered to operate in two major modes: Standard UI mode and Extended Interface mode. In addition there is the Sleep state, which can overlay the major modes in specific circumstances. For more information on iPod power states, see *iPod/iPhone Hardware Specifications*.

Standard UI Mode is the user interface mode that allows the iPod to be driven by its front panel display and buttons. Alternatively, the iPod may be placed in Extended Interface mode and controlled by an accessory device, using the Extended Interface protocol and its lingo (Lingo 0x04) as described in this document. The present chapter briefly summarizes the iPod Extended Interface and Sleep modes.

**Note:** The iPhone and iPhone 3G do not have a Sleep state.

Accessories should query the iPod for the Extended Interface protocol version number, by sending the General lingo command `GetLingoProtocolVersion`, to determine which Extended Interface features are supported by the iPod. The Extended Interface protocol version number is incremented with the addition of new command IDs or with bug fixes to existing iPod features.

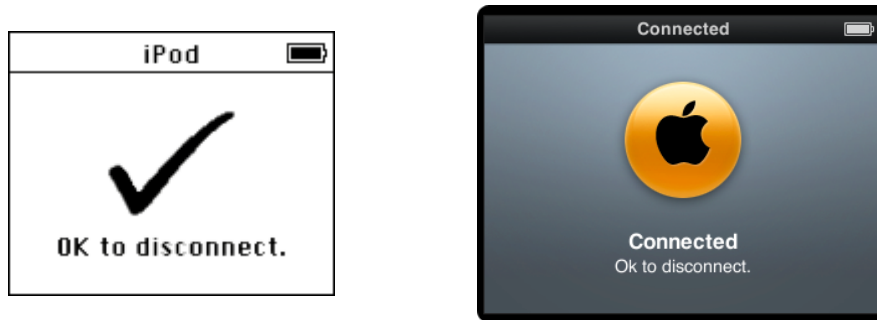
## Extended Interface Mode

---

The iPod transitions into the Extended Interface mode when a device is connected to it and issues an `EnterRemoteUIMode` command. The transition to Extended Interface mode is described in ["Accessory Identification and Authentication"](#) (page 344).

If the iPod is playing an audio track during this transition, the playback is automatically paused; video tracks are stopped. By default, the iPod's display changes to an "OK to disconnect" screen such as those shown in [Figure 4-1](#) (page 342).

**Figure 4-1** Typical "OK to disconnect" screens



The Extended Interface protocol allows accessories to replace the checkmark graphic with a downloaded image set through ["Command 0x0032: SetDisplayImage"](#) (page 416). Removing power from the iPod while a connection remains results in the iPod going into a Sleep state after two minutes of inactivity. The keys and wheel on the front of the iPod are disabled when in Extended Interface mode.

The iPod transitions back to Standard UI mode when any of the following occurs:

- The device issues an `ExitRemoteUIMode` command.
- The device reidentifies itself, using the `StartIDPS` command (see ["Device Signaling and Initialization"](#) (page 48)).
- The device is disconnected from the iPod.

If the iPod is playing a track during this transition, the playback is automatically paused. Any iPod settings with the restore on exit feature set are restored when the iPod is disconnected.

## Sleep State

---

The iPod screen, playback, and most major parts of the iPod are off while the iPod is in the Sleep state. The iPod transitions from Extended Interface mode to the Sleep state when power is detached and playback is idle. A two minute period of inactivity is required before the iPod transitions into the Sleep state. When power is restored, the iPod returns to the Extended Interface mode.

An iPod will not sleep while it remains attached to an active USB host. The USB host must switch off its host controller to force an iPod in Extended Interface mode into the Sleep state. If the iPod is not currently in Extended Interface mode, playback must be paused before the host controller is turned off. In Extended

Interface mode, there is no need to pause iPod playback before turning off the host controller because this action generates a disconnect event that causes the iPod to exit the Extended Interface mode and allows the iPod to transition into the Sleep state. Attaching USB power to an iPod in Sleep state will wake it up.

Refer to the *iPod/iPhone Hardware Specifications* for information about transitions to the Sleep state from Standard UI mode.

## Packet Formats

The Extended Interface packet format is compatible with the packet formats described in "[Command Packet Formats](#)" (page 58). The primary difference between the two is the use of a 16-bit command ID field in Lingo 0x04 packets versus an 8-bit command ID field in other packet-based lingo. Depending on the length of the packet, either the small packet format (payloads of 252 bytes or less) or the large packet format (253–65532 byte payloads) will be used.

**Note:** The Extended Interface start of packet (SOP) byte has the same value as the start of packet byte used in other iAP packets.

### Small Packet Format

For packets whose payloads are 252 bytes or less, use the small packet format. The small packet format is shown in [Table 4-1](#) (page 343).

**Table 4-1** Small packet format

Byte number	Value	Meaning
0x00	0xFF	Sync byte (required only for UART serial)
0x01	0x55	Start of packet
0x02	0xNN	Packet payload length
0x03	0x04	Lingo ID: Extended Interface lingo
0x04	0xNN	Command ID (bits 15:8)
0x05	0xNN	Command ID (bits 7:0)
0x06...0xNN	0xNN	Packet command data (0–252 bytes)
(last byte)	0xNN	Packet payload checksum byte

### Large Packet Format

For packets whose payloads are between 253 bytes and 65532 bytes in length, use the large packet format. The large packet format is shown in [Table 4-2](#) (page 344).

**Table 4-2** Large packet format

Byte number	Value	Meaning
0x00	0xFF	Sync byte (required only for UART serial)
0x01	0x55	Start of packet
0x02	0x00	Packet payload length marker
0x03	0xNN	Packet payload length (bits 15:0)
0x04	0xNN	Packet payload length (bits 7:0)
0x05	0x04	Lingo ID: Extended Interface lingo
0x06	0xNN	Command ID (bits 15:8)
0x07	0xNN	Command ID (bits 7:0)
0x08...0xNN	0xNN	Packet command data (0–65532 bytes)
(last byte)	0xNN	Packet payload checksum byte

## Packet Details

The sync byte (0xFF) is not considered part of the packet. It is sent merely to facilitate automatic baud rate detection and correction when communicating over the UART serial port link. The sync byte is required only when communicating over the UART serial port link. It must not be sent when communicating over the USB port link. The packet payload length is the number of bytes in the packet not including the sync byte, packet start byte, packet payload length byte or bytes, or packet payload checksum. That is, it is the length of the lingo ID, the command ID, and the command data, if any. The lingo ID specifies the category this communication falls under; in this case, it is the iPod Extended Interface (0x04). The command ID is a 16-bit value, as compared with 8-bit command ID used by many other lingoes, that gives a specific indication of the packet's purpose.

The sum of all the bytes from the packet payload length (or length marker, if applicable) through the packet payload checksum is 0. The checksum must be calculated appropriately, by adding the bytes together as signed 8-bit values, discarding any signed 8-bit overflow, and negating the sum to create the signed 8-bit checksum byte.

## Accessory Identification and Authentication

When attached, an accessory must detect the iPod and then identify and authenticate itself, as described in ["Device Signaling and Initialization"](#) (page 48). To do this, the accessory must send a `StartIDPS` command to begin the Identify Device Preferences and Settings (IDPS) process.

As part of its identification and authentication, the accessory should query the iPod's support for Extended Interface mode by sending a `GetiPodOptionsForLingo` command for Lingo 0x04. The 64-bit field in the `RetiPodOptionsForLingo` reply declares which Extended Interface features the iPod supports, as listed in [Table 4-3](#) (page 345).

**Table 4-3** `RetiPodOptionsForLingo` option bits for Lingo 0x04

Bit	Description
0x00	Video browsing
0x01	Remote UI enhancements
0x02	Nested playlists
0x03	Reserved
0x04	Supports Set Display Image
0x05-0x3F	Reserved

A simplified sequence of iAP General lingo commands for accessory identification and authentication is shown in [Table 4-4](#) (page 345). These commands identify the accessory, determine which options the attached iPod supports, set the accessory's preferences in the iPod, and authenticate the accessory.

**Table 4-4** Simplified IDPS and authentication command sequence

Step	Accessory command	iPod command	Comment
1	<code>StartIDPS</code>		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to <code>StartIDPS</code> returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> <li>■ If the accessory receives no ACK command within 1 second, it may retry Step 1 at 1-second intervals as long as iPod Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.</li> <li>■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the iPod does not support IDPS. Within 800 ms the accessory must send an <code>IdentifyDeviceLingoes</code> command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0.</li> <li>■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.</li> </ul>			
3	<code>GetiPodOptions-ForLingo</code>		getting options for General Lingo
4		<code>RetiPodOptions-ForLingo</code>	returning options for General Lingo

Step	Accessory command	iPod command	Comment
5	GetiPodOptions-ForLingo		getting options for Extended Interface Lingo
6		RetiPodOptions-ForLingo	returning options for Extended Interface Lingo
7	SetFIDTokenValues		setting full ID (FID) tokens that identify accessory preferences
8		RetFIDTokenValueACKs	acknowledging FID tokens that identify accessory preferences
Repeat Steps 7 and 8 until all required accessory preferences have been set and acknowledged			
9	EndIDPS		status 'finished with IDPS; proceed to authentication'
10		IDPSStatus	status 'ready for auth'
11		GetDevAuthentication-Info	no params
12	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1; cert data: ...
13		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
14	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1; cert data: ...
15		AckDevAuthentication-Info	acknowledging 'auth info supported'
16	EnterRemoteUIMode		entering Extended Interface mode
17		ACK	acknowledging 'Command Pending' to command 'General Lingo::EnterRemoteUIMode'
18		ACK	acknowledging 'Success (OK)' to command 'General Lingo::EnterRemoteUIMode'
19		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
20	RetDevAuthentication-Signature		returning signature '...'
21		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'

**Note:** The process of device identification is different for accessories that must support the 3G iPod. See “Interfacing With the 3G iPod” in *iPod/iPhone Hardware Specifications*.

## Command Transports

To establish reliable communication over UART transport, it may be necessary to repeat the device identification sequence multiple times. There can be problems when the iPod tries to synchronize with the device's baud rate, resulting in packet loss. Repeating the device identification sequence until it is successful ensures that iPod identifies the device correctly. This problem does not occur with USB transport.

Devices using a UART serial port link should send an extra sync byte before a command packet in order to wake up a sleeping iPod. This recommendation does not apply while the iPod is in Extended Interface mode because iPods do not sleep in this mode as long as external power is supplied by an accessory device. If the device allows the iPod to transition to Sleep state by removing external power, it should send extra sync bytes when waking up the iPod. For further information about supported transport protocol links, see *iPod/iPhone Hardware Specifications*.

## Entering Extended Interface Mode

The iPod does not automatically enter into Extended Interface mode; the device must explicitly switch into and out of the Extended Interface mode. To confirm that the iPod has gone into Extended Interface mode, the device sends it the Lingo 0x04 command `GetPlayStatus`. Receiving a valid `ReturnPlayStatus` command tells the device that the iPod has entered Extended Interface mode and Lingo 0x04 commands can be used. The iPod will return Lingo 0x04 ACK commands with a bad parameter status if the device is not communicating properly with the iPod in Extended Interface mode.

Accessories may send Extended Interface command packets only after they have successfully identified and switched into Extended Interface mode. Accessories must still obey the other inter-packet and inter-command timing issues specified in “[Command Timings](#)” (page 146).

## Using the Extended Interface Protocol

There are four General lingo (Lingo 0x00) commands that allow devices to determine what mode the iPod is in and to switch between the two major modes, Standard UI and Extended Interface. These commands were implemented to allow a device to switch between modes without having to unplug the device. Multilingo devices must use these commands to switch into and out of the Extended Interface mode.

**Note:** Multilingo devices that support the Extended Interface do not automatically switch into the Extended Interface mode after the identification process completes. These devices must use the General lingo mode switching commands to explicitly switch into Extended Interface mode.

[Figure 4-4](#) (page 362) lists the General lingo command codes for querying, entering, and exiting the Extended Interface protocol.

**Table 4-5** General lingo commands for switching modes

Command ID	General lingo command	Protocol version	Requires authentication	
			UART serial port link	USB port link
0x03	RequestRemoteUIMode	1.00	No	Yes
0x04	ReturnRemoteUIMode	1.00	No	Yes
0x05	EnterRemoteUIMode	1.00	No	Yes
0x06	ExitRemoteUIMode	1.00	No	Yes

Two logical entities need to be managed while browsing and playing content in Extended Interface mode: the content Database Engine and the Playback Engine. The following sections describe those engines and give an example of command traffic between a device in Extended Interface mode and an iPod.

[Table 4-10](#) (page 368) lists the complete set of iPod Extended Interface command IDs, noting which are relevant to the Database and Playback Engines, as well as which the protocol versions in which those commands were introduced. Unless otherwise noted, all subsequent protocol versions will support existing commands.

## The Playback Engine

The Playback Engine is active when the iPod is in a playback state, such as play, fast forward, and rewind. It has a special play list, called the Now Playing playlist, that is used to determine what track or content item will be played next. The `PlayCurrentSelection` command is commonly used to transfer the currently selected database items to the Now Playing playlist and start the player at a specified item within that list. The `SelectDBRecord` and `SelectSortDBRecord` commands, with a track or audiobook category and a valid index, also transfer selected database items to the Now Playing playlist. Changes to the database selection before or after these commands have no effect on the current playback.

### Playback Behavior

An iPod's behavior when it receives a `PlayCurrentSelection` packet may depend on the shuffle setting, the repeat setting, and on the kind of tracks in a playlist. Consider a playlist with the following tracks:

Track A  
Track B  
Track C  
Track D  
Track E

If repeat is off and shuffle mode has been turned on, using `SetShuffle`, a `PlayCurrentSelection` packet with an index of -1 causes the playlist to be shuffled randomly and sent to the Playback Engine. Play starts at the first track of the new order. For example, the playlist may be sent to the Playback Engine in this order:

Track D  
Track A



Track B  
Track E  
Track C

Playback starts at Track D, proceeds to Track A, and so on.

If repeat is off and shuffle mode is off, a `PlayCurrentSelection` packet with an index of 0x00000000 causes the playlist to be sent to the Playback Engine in the current playlist order, with play starting at the first track. Using the foregoing example, the playlist is sent to the Playback Engine in its original order:

Track A  
Track B  
Track C  
Track D  
Track E

Playback starts at Track A, proceeds to Track B, and so on.

If both repeat and shuffle mode are on, a `PlayCurrentSelection` packet with an index of 0 or greater causes the playlist to be shuffled randomly but with the first track set to the selected index. For example, with an index of 4, the playlist may be sent to the Playback Engine in the following order and play starts with Track E. If repeat is set to One, track E will repeat; if it is set to All, the entire playlist will repeat.

Track E  
Track B  
Track A  
Track D  
Track C

**Note:** The index numbers of tracks in the Playback Engine are always arranged in the current playlist order, starting with 0.

If repeat and shuffle mode are both off, a `PlayCurrentSelection` packet with an index of 0 or greater causes the playlist to be sent to the Playback Engine in the current playlist order but with the first track set to the passed index. With an index of 2, using the current example, the playlist is sent to the Playback Engine in this order but play starts at Track C, proceeds to Track D, and so on:

Track A  
Track B  
Track C  
Track D  
Track E

On clickwheel iPods (without touch screens) certain sequences of playback requests can result in an empty playlist. For example, consider the following sequence of packets:

```
SetShuffle(on)
ResetDBSelection()
SelectDBRecord(playlist)
PlayCurrentSelection(-1)
```

If the selected playlist contains only tracks that have the Skip When Shuffle attribute (for example, a playlist of audiobooks) then the playlist that is copied to the Playback Engine will be empty and no tracks will play. If `PlayCurrentSelection(0)` were sent for the last packet instead of `PlayCurrentSelection(-1)`, the track with index 0 would play but no other tracks would play after it finished.

In all the foregoing scenarios, the iPod returns a successful `ACK(0)` packet to the accessory. The accessory should not assume that any tracks are playing, but should use `GetPlayStatus` to verify that playback has started.

**Note:** Commands in other lingoes may change the iPod's playback status, including its player state, track position, and track index. Examples include the Simple Remote lingo command `ContextButtonStatus` and the Display Remote lingo command `SetCurrentPlayingTrack`. Accessories should request notifications (via `SetPlayStatusChangeNotification`) or repeatedly send `GetPlayStatus` to coordinate itself with the iPod's playback status.

## The Database Engine

The Database Engine is always accessible when the unit is awake. It can be manipulated remotely and allows groups of content items to be selected, independently of the Playback Engine. This allows the user to listen to an existing track or playlist while checking the iPod database for another selection. Once a different database selection is made, the user selection (the track or content playlist) is sent to the Playback Engine. The commands such as `ResetDBSelection` and `GetNumberCategorizedDBRecords` are examples of commands that are used to manipulate the Database Engine. You can identify database commands from playback commands by the string "DB" or "Database" embedded in the command name.

### Database Category Hierarchies

The database engine uses **categories** to classify music, videos and other records stored in the database. Possible categories are playlist, genre, media kind, artist, album, track, composer, audiobook, and podcast. A list of records can be assembled, based on the various selected categories, to create a user list of records (a playlist).

The database categories have a hierarchy by which records are sorted and retrieved. This category hierarchy has an impact on the order in which records must be selected. For example, if a low category, such as album, is selected first, followed by a higher relative category such as genre, the album selection is invalidated and is ignored. When creating a new set of database selections, the device must begin by resetting all database selections, using the `ResetDBSelection` or `ResetDBSelectionHierarchy` commands, and selecting the desired database categories from highest to lowest relative category. The category hierarchy, from highest to lowest (excluding podcasts), is shown in [Table 4-6](#) (page 350).

**Table 4-6** Database category hierarchy (excluding podcasts)

Category	Notes
All (highest level)	This is the state after a <code>ResetDBSelection</code> or <code>ResetDBSelectionHierarchy</code> command. No database categories are selected. If the <code>GetNumberCategorizedDBRecords</code> command is sent while in this state, it returns the total number of records for the requested category.

Category	Notes
Playlist	When the <code>SelectDBRecord</code> command selects a playlist, all lower database category selections (genre or media kind, artist or composer, album, and track or audiobook) are invalidated.
Genre or Media Kind	When the <code>SelectDBRecord</code> command selects a genre, all lower database category selections (artist or composer, album, and track) are invalidated.
Artist or Composer	When the <code>SelectDBRecord</code> command selects an artist or composer, all album and track category selections are invalidated.
Album	When the <code>SelectDBRecord</code> command selects an album, all track category selections are invalidated.
Track or Audiobook (lowest level)	When the <code>SelectDBRecord</code> command selects a track (music or video) or an audiobook, it is automatically transferred from the Database Engine to the Playback Engine.

There is a parallel hierarchy for podcasts. When the podcast category is selected, all track or audiobook selections are invalidated. If other categories (such as Artist) are then selected after the podcast category, the list of podcasts returned by `GetNumberCategorizedDBRecords` and `RetrieveCategorizedDBRecords` is affected. This behavior parallels that of playlists with multiple categories selected.

The default sort order for podcasts is alphabetical. For episode tracks, the order is reverse chronological; that is, the most recent one is first.

**Table 4-7** Database category hierarchy for podcasts

Category	Notes
All (highest level)	This is the state after a <code>ResetDBSelection</code> or <code>ResetDBSelectionHierarchy</code> command. No database categories are selected. If the <code>GetNumberCategorizedDBRecords</code> command is sent while in this state, it returns the total number of records for the requested category.
Podcast	When the <code>SelectDBRecord</code> command selects a podcast, all lower database category selections are invalidated.
Episode Track (lowest level)	When used in the context of podcasts, episode is synonymous with track. When the <code>SelectDBRecord</code> command selects an episode track, it is automatically transferred from the Database Engine to the Playback Engine.

## Database Category Counts

The record count for a given database category is established either when the `GetNumberCategorizedDBRecords` command, or when the `SelectDBRecord` command is processed for that category. Depending on the previous database state and the order in which the categories are selected, the category record counts returned by the `GetNumberCategorizedDBRecords` command may be different.

After all database selections have been reset using the `ResetDBSelection` command, sending the `GetNumberCategorizedDBRecords` command for any category retrieves the total number of records matching that category in the entire iPod database. In contrast, if one or more categories are already selected, the `GetNumberCategorizedDBRecords` command returns the number of records that match the both the specified category, as well as any already selected categories. As an example, assume an iPod has the following four tracks in its database:

Genre	Artist	Album	Track
Electronica/Dance	Dirty Vegas	Essential Mix	Days Go By
R&B	Radiance	Pick 'n Choose	All Night
Hip-Hop/Rap	Coolio	It Takes A Thief	Fantastic Voyage
Electronica/Dance	New Order	Power, Corruption & Lies	Blue Monday

Imagine the following command sequence:

1. The database is reset using `ResetDBSelection`. All record counts are cleared.
2. The track record count returned by `GetNumberCategorizedDBRecords` is four, the total number in the database.
3. The genre Electronica/Dance is selected. The track record count returned by `GetNumberCategorizedDBRecords` is two. Two records match the Electronica/Dance genre category: the tracks by Dirty Vegas and New Order.
4. The artist Dirty Vegas is selected. Because the genre category is still selected, the track record count returned by `GetNumberCategorizedDBRecords` is one. One record matches both the Electronica/Dance genre category and the Dirty Vegas artist category: the track "Days Go By."

Note that in this example, each time a category is selected the track count is invalidated because it is in a lower category than the category selected. See ["Database Category Hierarchies"](#) (page 350) for details about the effects a category selection has on other category selections.

Retrieving the record count of a higher category does not cause the current selection to be invalidated. However, selecting one of the results of that count does cause an invalidation. For example, using the same iPod database, a different command sequence produces different results:

1. The database is reset using `ResetDBSelection`.
2. The artist Dirty Vegas is selected. The Track record count returned by `GetNumberCategorizedDBRecords` for the Artist category is one, as there is only one track matching the Dirty Vegas artist category: the track "Days Go By."
3. The Genre record count is requested using `GetNumberCategorizedDBRecords`; this also returns a record count of one. Had the database included multiple Dirty Vegas songs from the same genre, then the returned Genre record count would still have been one. Had the database included Dirty Vegas songs from multiple different genres, however, the returned Genre record count would have been greater than one.

4. The Electronica/Dance genre is selected, which results in the Dirty Vegas selection being invalidated. If the `GetNumberCategorizedDBRecords` command is used to obtain the record count for the Artist category, it returns two, the expected total number of artists matching the Electronica/Dance genre (Dirty Vegas and New Order).

## The All Tracks and On-The-Go Playlists

---

The iPod has two playlists that developers should be aware of: the All Tracks and On-The-Go playlists. The All Tracks playlist contains every track in the current hierarchy (either audio or video) that is stored on the iPod and always resides at index 0x00 in the Playlist category. The On-The-Go playlist is always stored at the last index in the audio playlist category and the user controls whether it contains playlist data. It is not supported under video browsing.

To populate the On-The-Go playlist, a user must select an audio track, artist, or album by locating the item in the iPod menu hierarchy and holding the select button down for several seconds. The track, album, or artist name will blink for a few seconds indicating that the track or tracks have been added to the On-The-Go playlist. Once the tracks have been added to the On-The-Go playlist, the user may clear the playlist through the playlist menu.

Accessories can use the following steps to determine whether there are any tracks available in the On-The-Go playlist:

1. Call `ResetDBSelection()`
2. Call `GetNumberCategorizedDatabaseRecords(playlists)`. This returns the number of playlists available on the iPod (the playlist count).
3. Call `SelectDBRecord(playlists, playlistCount - 1)`. This selects the On-The-Go playlist which is always the last entry in the playlist list.
4. Once the On-The-Go playlist is selected, the accessory can query the database for the number of available records using the `GetNumberCategorizedDBRecords` call with the category Tracks.

Accessories can use the following steps to select and play all the tracks on the iPod:

1. Call `ResetDBSelection()`
2. Call `GetNumberCategorizedDatabaseRecords(playlists)`. This returns the number of playlists available on the iPod.
3. Call `SelectDBRecord(playlists, 0)`. This selects the All Tracks playlist.
4. Call `PlayCurrentSelection()`.

## Nested Playlists

---

Version 1.13 and later of the iPod Extended Interface protocol supports nested playlists. A playlist can contain either tracks or one or more other playlists, but not both. The number of playlist nesting levels is unlimited.

When navigating playlists using the iPod's Playlist category, nested playlists are flattened and their contents are presented in the highest level playlist that contains them. For example, if playlist Adam contains two nested playlists, Bob and Carol, then the Playlist category displays three playlists: Adam, Bob, and Carol. The contents of playlists Bob and Carol are their tracks. The contents of playlist Adam is the concatenation of the tracks in playlists Bob and Carol.

The iPod provides a Nested Playlist category to traverse playlists with their nesting hierarchy in place. In the foregoing example, only playlist Adam would be returned when traversing the Nested Playlist category after a `ResetDBSelection` command. Selecting playlist Adam would expose nested playlists Bob and Carol. Selecting Bob or Carol would expose no further nested playlists. A typical command sequence for traversing the hierarchy looks like this:

1. Call `ResetDBSelection`; all record counts are cleared.
2. Call `GetNumberCategorizedDBRecords`; a `NestedPlaylist` count of 1 is returned.
3. Call `SelectDBRecord`; select a `NestedPlaylist` index of 0 (playlist Adam).
4. Call `GetNumberCategorizedDBRecords`; a `NestedPlaylist` count of 2 is returned.
5. Call `SelectDBRecord`; select a `NestedPlaylist` index of 1 (playlist Carol).
6. Call `GetNumberCategorizedDBRecords`; a `NestedPlaylist` count of 0 is returned.

When the count of nested playlists returned by `GetNumberCategorizedDBRecords` is 0, the current playlist has no nested playlists. At this point the accessory should query the iPod for the number and names of the tracks in that playlist.

Calling `SelectDBRecord` with a `NestedPlaylist` index of -1 traverses the hierarchy upward to the next parent playlist. If the current playlist has no parent playlist, passing index -1 has no effect.

---

## Unexpected Database Changes

Wireless iTunes store access, rentals, Genius playlists, On-The-Go playlists, and other media features present opportunities for the iPod's database contents to change without synchronizing to the iTunes database on a host computer. Category counts, category lists, track counts, tracks, and other media information may change at any time, including during Extended Interface sessions. Playback engine counts and contents may also change as media are added to or removed from the iPod.

Accessories must not expect the media information and content to remain the same throughout an Extended Interface session. Accessories must verify media information immediately before use to ensure that the data has not changed, especially when media information is cached locally on the accessory. Even then, the database or playback engine state may change without the iPod informing the accessory. Accessories using the Extended Interface must be programmed to recover from command errors and failures received unexpectedly from the iPod, by retrying commands or by reidentifying themselves and entering Extended Interface mode again.

---

## Transferring Album Art

The extended interface lingo includes several commands that support the transfer of album artwork from an iPod to an accessory:

- `GetArtworkFormats`
- `RetArtworkFormats`
- `GetTrackArtworkTimes`
- `RetTrackArtworkTimes`
- `GetTrackArtworkData`
- `RetTrackArtworkData`

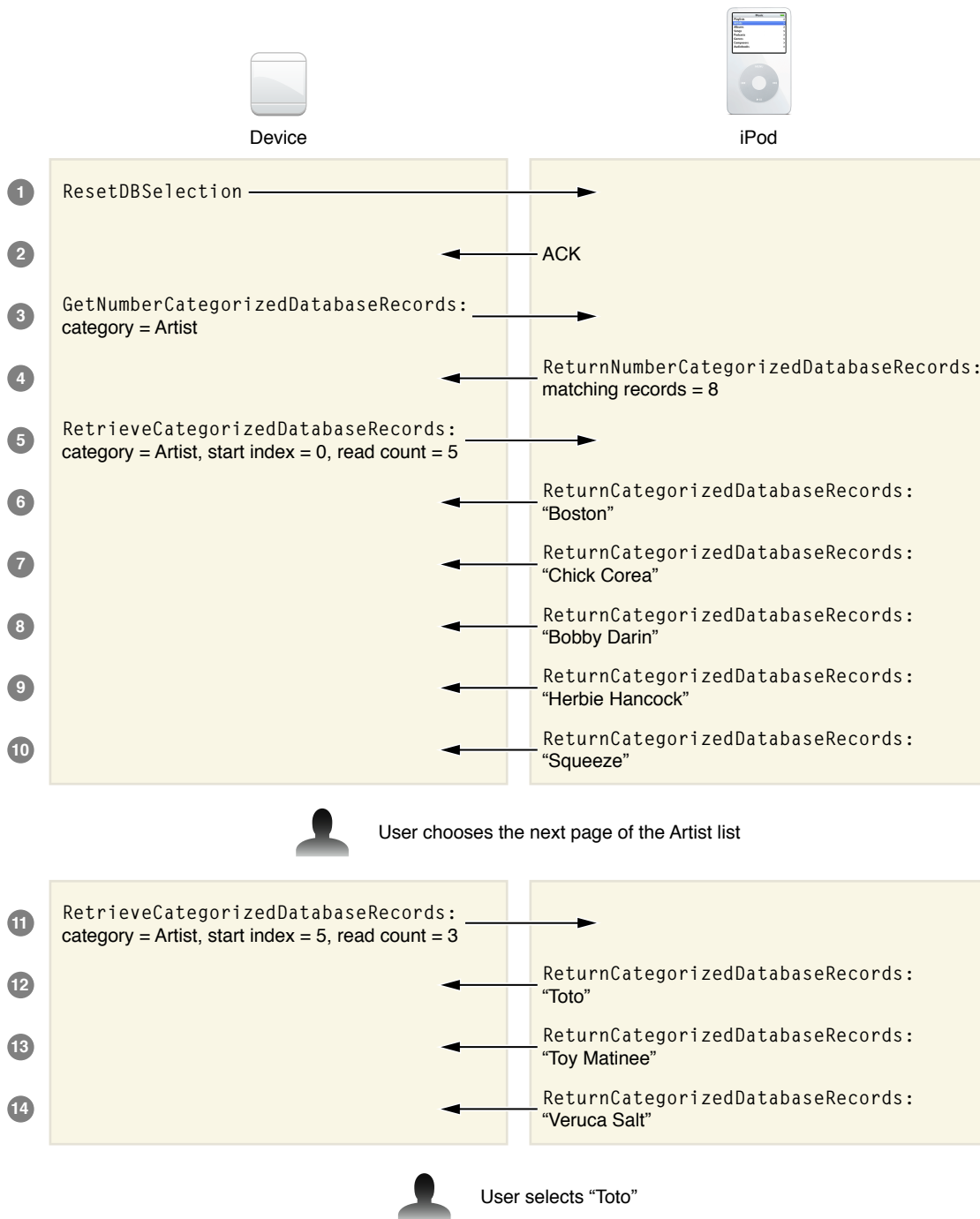
All album art image encoding is currently RGB-565, which can be transferred in both big- and little-endian formats. Artwork retrieval takes place in the following steps:

1. Retrieve the number of formats available for artwork on the iPod using `GetArtworkFormats`. It is not necessary to call `GetArtworkFormats` more than once per session; these values will be static while the accessory is attached to the iPod. However, there are no guarantees about the number of formats and which ones are available on a particular model or firmware version. Each `formatID` in `RetArtworkFormats` specifies both a pixel encoding, such as RGB-565 little-endian, and the image dimensions. All formats are fixed-size.
2. When the accessory wants to retrieve the artwork for a given track, it calls `GetIndexedPlayingTrackInfo` with an `infoType` of 7. This returns the count of artwork available for each `formatID` associated with the track. It is possible that a track may not have artwork for a particular `formatID` or that the number of images will vary by `formatID`. A given size of album artwork may be available for only one track in the database.
3. To retrieve the list of images associated with a given track and `formatID`, the accessory calls `GetTrackArtworkTimes`. This command tells the iPod to return the associated timestamp for each artwork. The timestamp indicates when the artwork should be displayed, expressed in milliseconds from the start of playback.
4. When the accessory wants to retrieve an individual piece of artwork, it sends `GetTrackArtworkData` to the iPod. This requires the accessory to specify a track, a `formatID`, and the timestamp of the desired image. The iPod returns the specified artwork and the accessory can display it whenever it chooses.

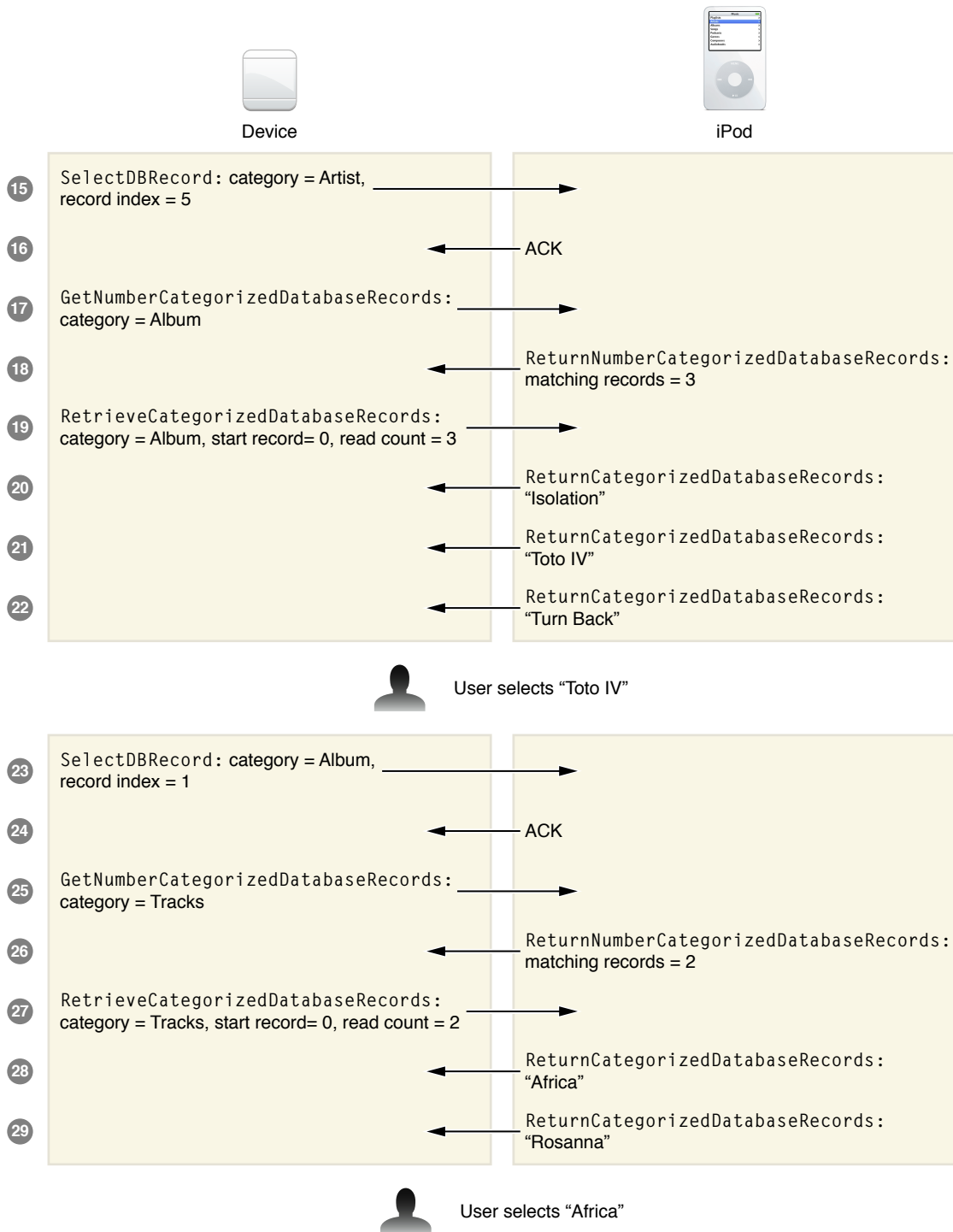
## Using the Extended Interface Protocol: An Example

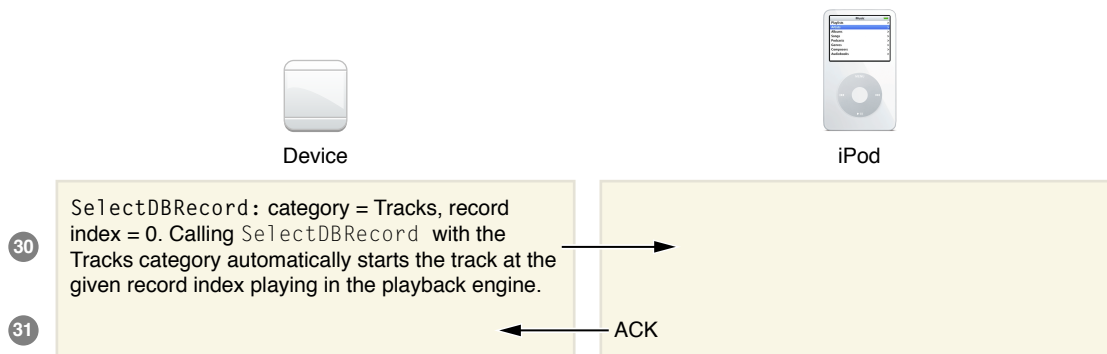
---

The following example of command traffic shows a conversation that might occur if the interfacing device has a paged menu of 5 choices; that is, a menu capable of displaying up to 5 options for the user to choose from.

**Figure 4-2** An example interaction between an Extended Interface device and an iPod







## Video Browsing

With iPod models that store and display video content (in addition to music), an accessory device may choose to navigate the iPod's hierarchy of stored videos. To do this, the accessory must use the command `ResetDBSelectionHierarchy` to switch from navigating music tracks to navigating video tracks.

### Video Playlists

A video playlist is a collection of tracks, created in iTunes, that contains one or more tracks playable as video. In contrast, an audio-only playlist does not contain any tracks playable as video. Video playlists may contain video-only tracks (such as movies) or tracks playable as either video or audio (such as video podcasts or music videos). Audio-only tracks are not visible and not playable when the iPod is navigating the video hierarchy.

### Video Navigation User Interface

Video track selection follows the iPod user interface rules as closely as possible and behaves in the same way as audio track selection. In the video hierarchy, the menu item `Genre` is used to indicate media kind. The list of media kinds is dynamic and may be updated in the future to add, modify, or remove entries.

Once a media kind has been selected, the existing `Artist`, `Album`, and `Song` or `Track` categories may be used to further narrow the selection. For some media kinds, such as movies, a category may contain a single entry. In such cases, the accessory may choose to bypass this menu level. For example, with TV shows the `Season` menu on the iPod is omitted if all the stored episodes are from the same season. Accessory designs that provide a video browsing user interface are encouraged to duplicate this behavior.

Video podcasts and music videos appear in both the music and video hierarchies. Other video tracks, such as movies and TV shows, appear only in the video hierarchy. Note that video playlists may contain audio-only tracks.

The `Artist` and `Album` categories do not apply to every media kind. For example, movies do not use these categories at the current time. When a category does not apply, the iPod returns a count of 1 and displays a localized version of the word "All." The accessory may choose to bypass this menu level when displaying it to the user, as described above.

## Navigating Video Playlists

---

The `ResetDBSelectionHierarchy` command with the video hierarchy type (0x02) selects the video hierarchy and invalidates all previous video database selections, such as `ResetDBSelection` in the audio-only hierarchy. As with the audio hierarchy, a video playlist record index of 0x00 designates the all video tracks playlist. If there are no video tracks on an iPod, the all video tracks playlist will still be present, with a record count of 0. In contrast to the audio hierarchy, there is no On-The-Go (OTG) playlist in the video hierarchy.

After a `ResetDBSelectionHierarchy` command is sent to select the video hierarchy, all video playlists become visible. The video playlist count can be obtained using the `GetNumberCategorizedDBRecords` command with the playlist category (0x01). The video playlist records can be obtained using the `RetrieveCategorizedDatabaseRecords` command. A specific video playlist can be selected using the `SelectDBRecord` command.

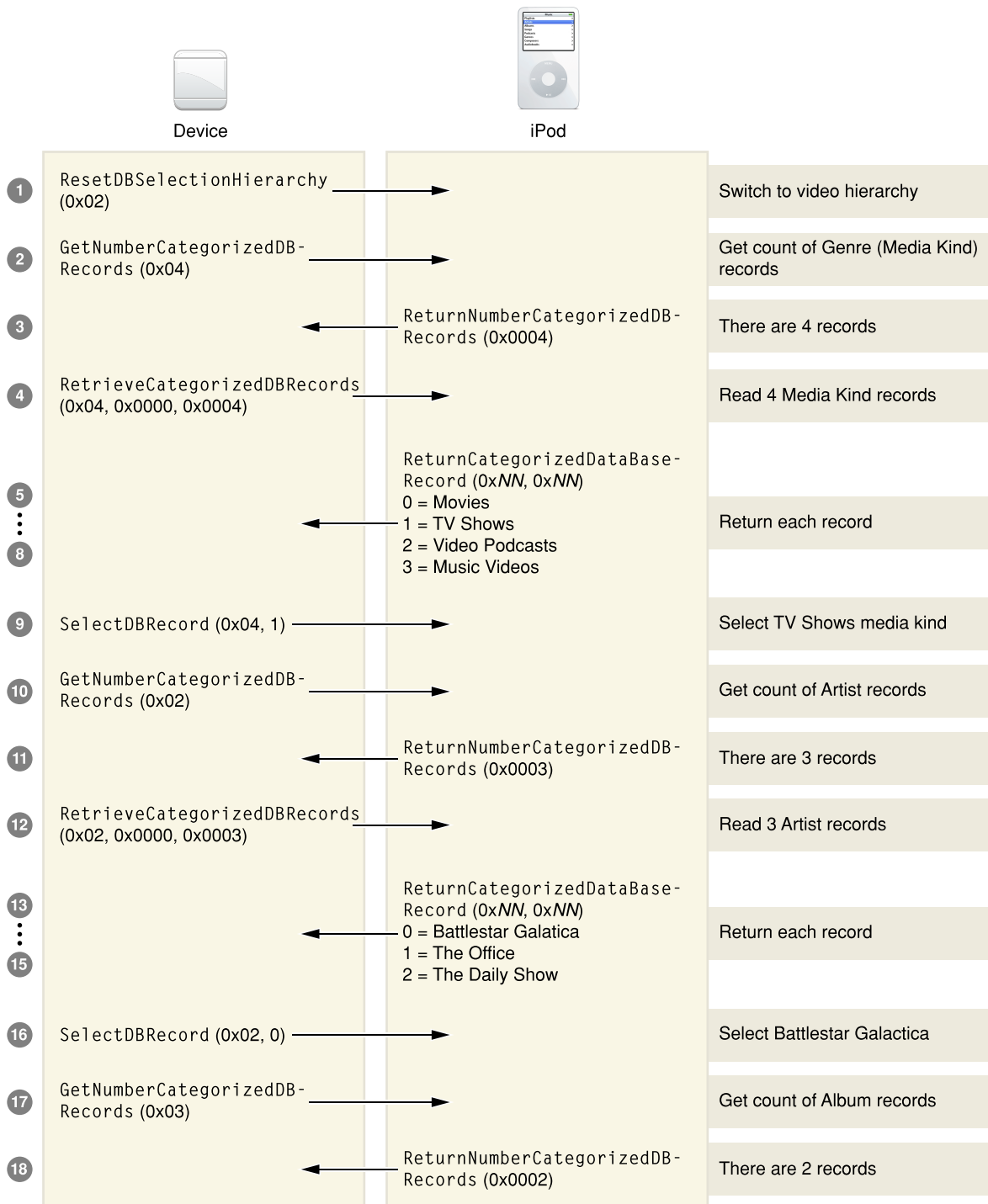
**Compatibility Note:** 5G iPods with firmware version 1.2.1 and earlier (before 11/2007) require that 1 be added to the `recIndex` values for music video artists. These iPods also do not filter out audio-only tracks while navigating the video hierarchy.

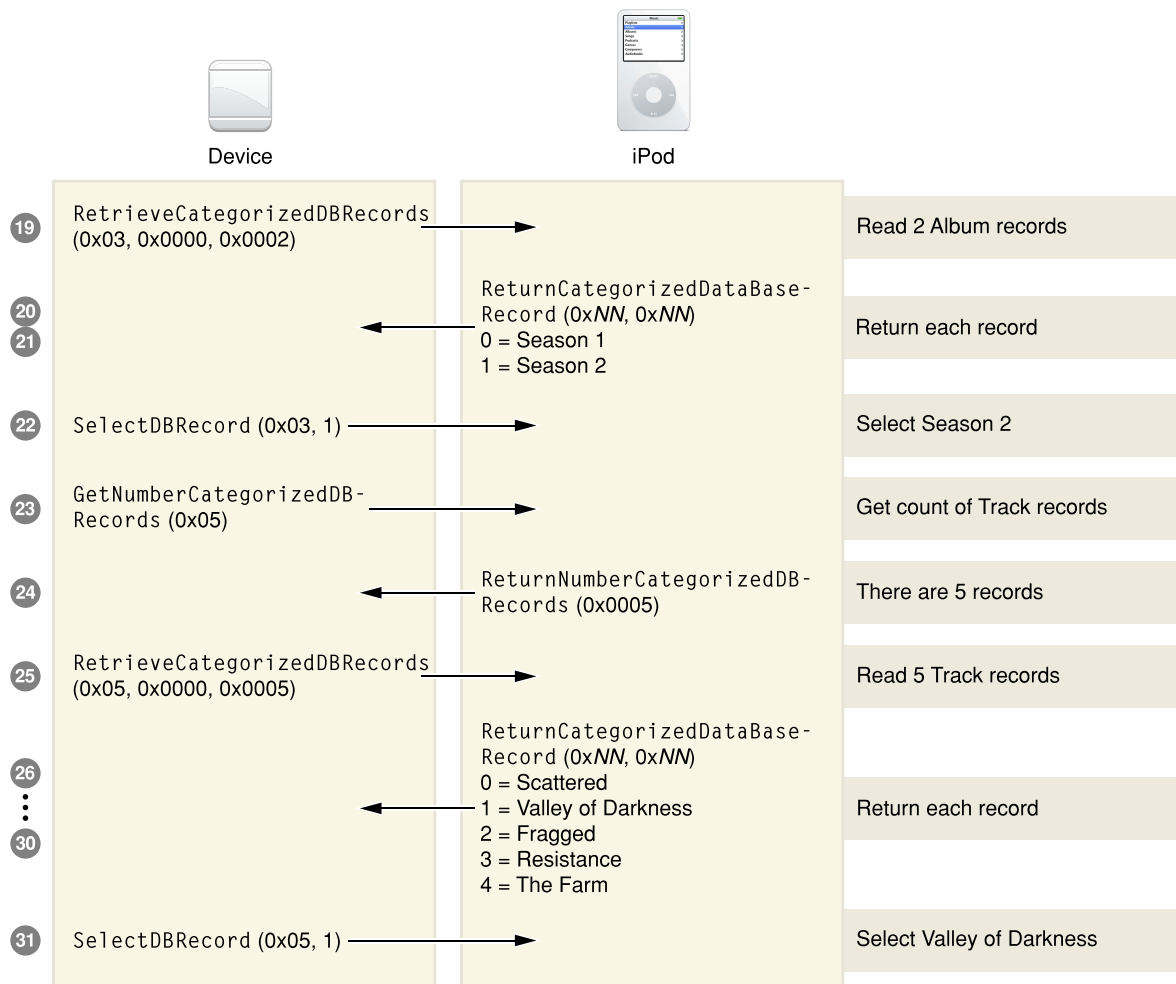
## Video Selection Examples

---

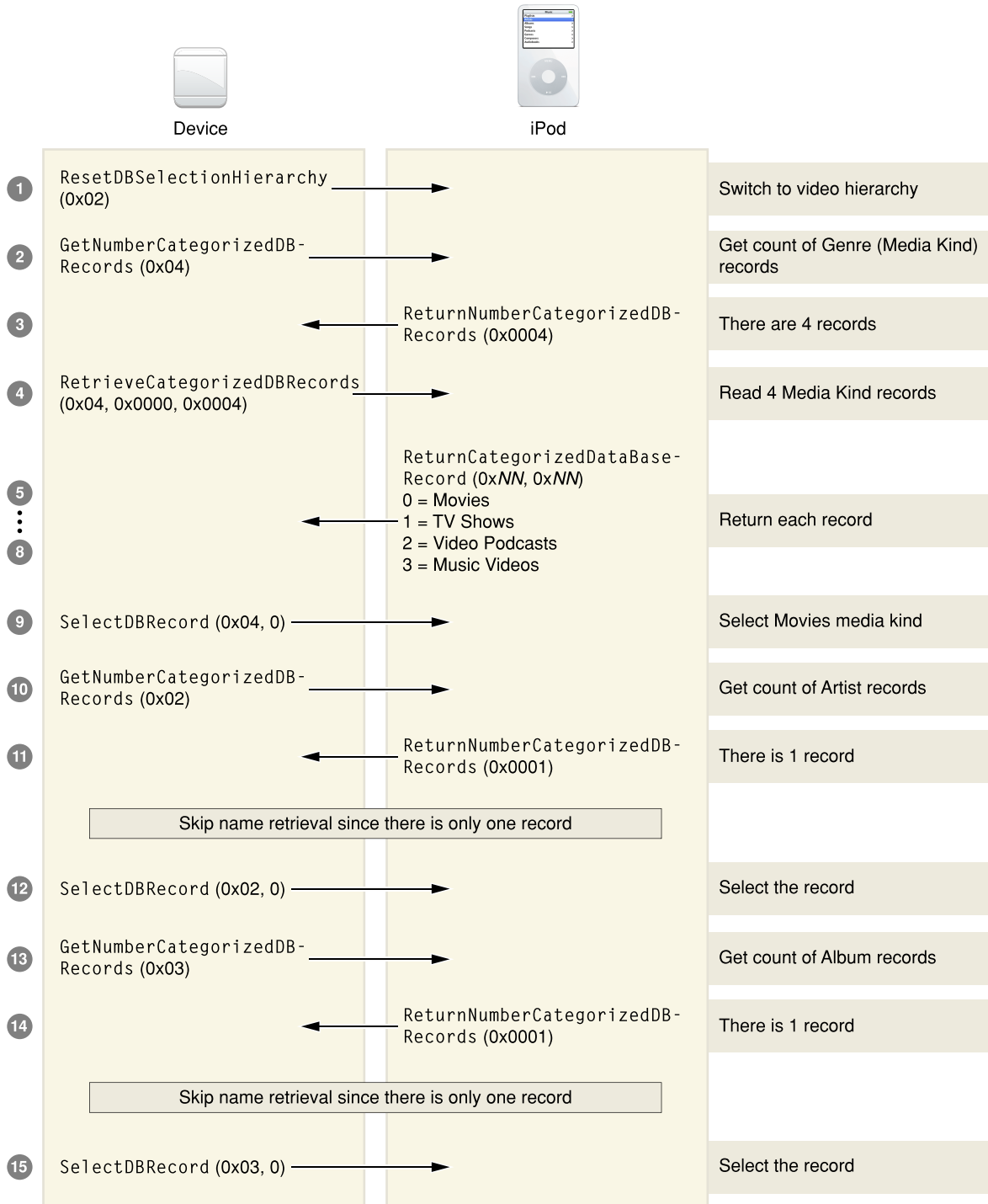
[Figure 4-3](#) (page 360) diagrams a typical interchange that might take place between an accessory and an iPod as the user of the accessory navigates to a TV show stored on an iPod.

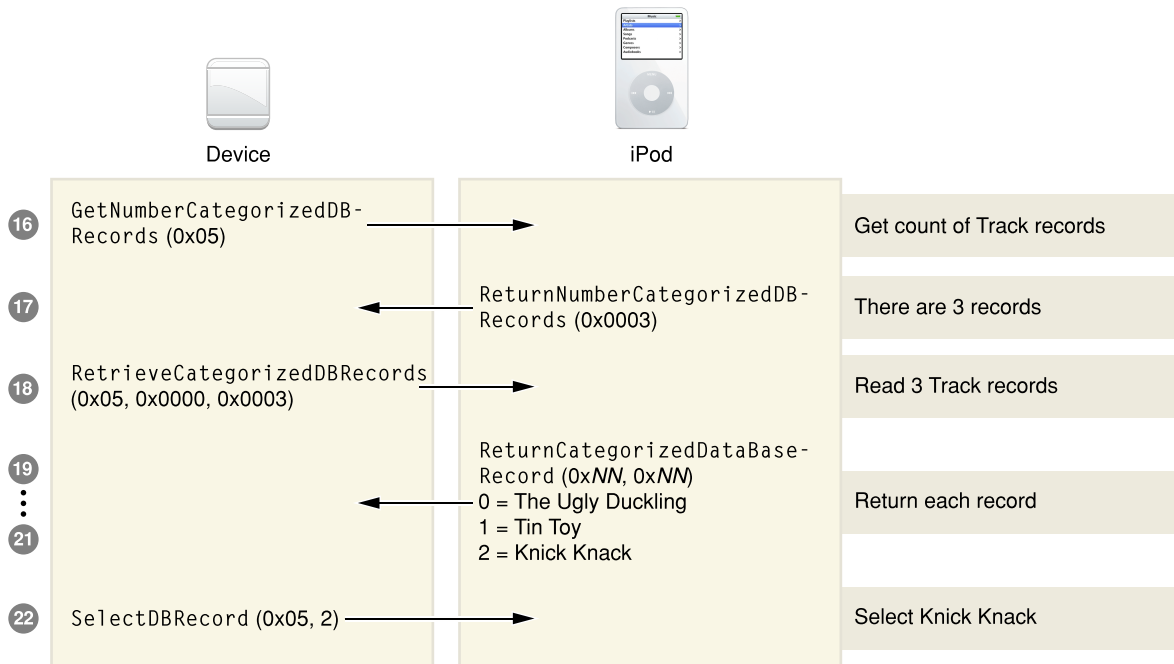
TV shows and video podcasts use both the Artist and Album categories. Music videos return “All” as the only Album entry. Movies return “All” for both Artist and Album categories. However, this behavior may change in future versions of the iPod firmware. Accessories must treat these categories as dynamic.

**Figure 4-3** Navigating to a TV show: example of interactions between a device and an iPod



**Figure 4-4** (page 362) diagrams the same kind of process with a movie. Note that in the case of the movie, the hierarchy search drops through two levels because those levels each contain only one record.

**Figure 4-4** Navigating to a movie: example of interactions between a device and an iPod



In video browsing, categorized DB records can be retrieved in any order, following essentially the same rules as navigating audio categories. It is not necessary to select the media kind. [Table 4-8](#) (page 363) illustrates the process of selecting a playlist that contains video material.

**Table 4-8** Selecting playlists containing video

Step	Action or command	Direction	Comments
1	Pass 0x02 (video) in <code>ResetDBSelection-Hierarchy</code>	Device to iPod	Switch to video hierarchy.
2	Pass 0x01 (playlist) in <code>GetNumberCategorizedDBRecords</code>	Device to iPod	Get count of playlists containing video. Any playlist that has at least one video track will be counted.
3	Receive 0x05 from <code>ReturnNumber-CategorizedDBRecords</code>	iPod to Device	There are 5 records.
4	Pass (0x01, 0x0000, 0x0005) in <code>RetrieveCategorized-DatabaseRecords</code>	Device to iPod	Get 5 video playlists starting with index 0.
5–9	Receive data from <code>ReturnCategorized-DataBaseRecord</code>	iPod to Device	Return 5 record strings from the database.
10	Pass (0x01, 0x0001) in <code>SelectDBRecord</code>	Device to iPod	Select video playlist index 1. Only video tracks will be displayed.

Following the steps shown above, the video track selection process can follow steps 16 to 22 shown in [Navigating to a movie: example of interactions between a device and an iPod](#) (page 363).

## Sample Extended Interface Session

Table 4-9 (page 364) shows a typical sequence of Extended Interface commands. This command sequence explores the iPod's database and then returns information about a specific track.

**Table 4-9** Typical extended interface commands

Step	Accessory command	iPod command	Comment
The device identifies itself, is authenticated by iPod, and enters extended interface mode.			
1	GetDBiTunesInfo(0x00)		Device requests database UID from iPod. The UID is used by the device to determine if this iPod database has been attached before.
2		RetDBiTunesInfo	The iPod returns database UID. The device checks the UID to see if it has a match to previously cached DB contents.
3	GetDBiTunesInfo(0x01)		The device requests database last sync date/time.
4		RetDBiTunesInfo	The iPod returns database last sync date/time. If DB has been cached by the device, the date/time is checked to see if the DB has been synced since the last time.
5	GetDBiTunesInfo(0x02)		The device requests database total audio track count.
6		RetDBiTunesInfo	The iPod returns database total audio track count. If DB has been cached by the device, the audio track count is checked to see if the DB has changed since the contents were cached. A count of 0 (no tracks are present) can be used to gray out or remove audio options in the accessory's user interface.
7	GetDBiTunesInfo(0x03)		The device requests database total video track count.



Step	Accessory command	iPod command	Comment
8		RetDBiTunesInfo	The iPod returns database total video track count. If DB has been cached by the device, the video track count is checked to see if the DB has changed since the contents were cached. A count of 0 (no tracks are present) can be used to gray out or remove video options in the accessory's user interface.
9	GetDBiTunesInfo(0x04)		The device requests database total audiobook track count.
10		RetDBiTunesInfo	The iPod returns database total audiobook track count. If DB has been cached by the device, the audiobook track count is checked to see if the DB has changed since the contents were cached.
11	ResetDBSelection		The device resets database selection before getting the total track count.
12		ACK	The iPod acknowledges the <code>ResetDBSelection</code> command.
13	GetNumberCategorized-DBRecords(0x05)		The device gets the database track count.
14		RetNumberCategorized-DBRecords	The iPod returns the database track count.
15	GetDBTrackInfo(0, trackCount, infoMask)		The device requests track information for all audio tracks. The <code>infoMask</code> tells iPod what types of information should be returned for each track.
16		RetDBTrackInfo(N, infoType, trackInfo)	The iPod returns track index N information of type <code>infoType</code> .
...			
17		RetDBTrackInfo(N, infoType+t, trackInfo)	The iPod returns track index N information of type <code>infoType+t</code> .
18		RetDBTrackInfo(N+1, infoType, trackInfo)	The iPod returns track index N+1 information of type <code>infoType</code> .
...			

Step	Accessory command	iPod command	Comment
19		RetDBTrackInfo(N+1, infoType+t, trackInfo)	The iPod returns track index N+1 information of type infoType+t.
...			
20		RetDBTrackInfo(N+n, infoType, trackInfo)	The iPod returns track index N+n information of type infoType.
...			
21		RetDBTrackInfo(N+n, infoType+t, trackInfo)	The iPod returns track index N+n information of type infoType+t.
22	GetNumPlayingTracks		The device gets the playing track count.
23		RetNumPlayingTracks	The iPod returns the playing track count.
24	GetPBTrackInfo(0, trackCount, infoMask)		The device requests track information for all audio tracks. The infoMask tells iPod what types of information should be returned for each track.
25		RetPBTrackInfo(N, infoType, trackInfo)	The iPod returns track index N information of type infoType.
...			
26		RetPBTrackInfo(N, infoType+t, trackInfo)	The iPod returns track index N information of type infoType+t.
27		RetPBTrackInfo(N+1, infoType, trackInfo)	The iPod returns track index N+1 information of type infoType.
...			
28		RetPBTrackInfo(N+1, infoType+t, trackInfo)	The iPod returns track index N+1 information of type infoType+t.
...			
29		RetPBTrackInfo(N+n, infoType, trackInfo)	The iPod returns track index N+n information of type infoType.
...			

Step	Accessory command	iPod command	Comment
30		RetPBTrackInfo(N+n, infoType+t, trackInfo)	The iPod returns track index N+n information of type infoType+t.
31	GetUIDTrackInfo(trackUID, infoMask)		The device requests track information for the specified track UID (the track UID was obtained via the GetDBTrackInfo or GetPBTrackInfo commands). The infoMask tells the iPod what types of information should be returned for the track.
32		RetUIDTrackInfo(UID, infoType, trackInfo)	The iPod returns track UID information of type infoType.
...			
33		RetUIDTrackInfo(UID, infoType+t, trackInfo)	The iPod returns track UID information of type infoType+t.

## Command Packets

This section describes the individual iPod Extended Interface (Lingo 0x04) command packets.

Unless otherwise specified, the following rules apply:

- All packet data fields larger than 8 bits are sent and received in big-endian format; that is, ordered from the most significant byte to the least significant byte.
- Device command packets that have a valid checksum but contain an invalid parameter, invalid command, or other such failure cause the iPod to respond with an ACK command containing the appropriate error status.
- A packet with an invalid checksum received by iPod is presumed to be invalid and is ignored. No ACK or other command is sent to the device in response to the invalid packet.
- All UTF-8 strings returned from the iPod are null-terminated.
- All commands require authentication before they can be used over the USB port link. Some commands require authentication when used over a serial link; see [Table 4-10](#) (page 368) for a list.

## Command Code Summary

[Table 4-10](#) (page 368) lists the valid command codes for the Extended Interface protocol.

**Table 4-10** Extended Interface lingo command summary

Command ID	Command	Target engine	Introduced in protocol version	Requires authentication over serial link
0x0000	Reserved	N/A	N/A	N/A
0x0001	ACK	N/A	1.00	No
0x0002	GetCurrentPlaying-TrackChapterInfo	Playback Engine	1.06	No
0x0003	ReturnCurrentPlaying-TrackChapterInfo	Playback Engine	1.06	No
0x0004	SetCurrentPlaying-TrackChapter	Playback Engine	1.06	No
0x0005	GetCurrentPlaying-TrackChapterPlayStatus	Playback Engine	1.06	No
0x0006	ReturnCurrentPlaying-TrackChapterPlayStatus	Playback Engine	1.06	No
0x0007	GetCurrentPlaying-TrackChapterName	Playback Engine	1.06	No
0x0008	ReturnCurrentPlaying-TrackChapterName	Playback Engine	1.06	No
0x0009	GetAudiobookSpeed	N/A	1.06	No
0x000A	ReturnAudiobookSpeed	N/A	1.06	No
0x000B	SetAudiobookSpeed	N/A	1.06	No
0x000C	GetIndexedPlaying-TrackInfo	Playback Engine	1.08	No
0x000D	ReturnIndexedPlaying-TrackInfo	N/A	1.08	No
0x000E	GetArtworkFormats	Playback Engine	1.10	No
0x000F	RetArtworkFormats	Playback Engine	1.10	No
0x0010	GetTrackArtworkData	Playback Engine	1.10	No
0x0011	RetTrackArtworkData	Playback Engine	1.10	No
0x0012	RequestProtocolVersion	See " <a href="#">Deprecated Extended Interface Commands</a> " (page 528).		

Command ID	Command	Target engine	Introduced in protocol version	Requires authentication over serial link
0x0013	ReturnProtocolVersion	See <a href="#">"Deprecated Extended Interface Commands"</a> (page 528).		
0x0014	RequestiPodName	See <a href="#">"Deprecated Extended Interface Commands"</a> (page 528).		
0x0015	ReturniPodName	See <a href="#">"Deprecated Extended Interface Commands"</a> (page 528).		
0x0016	ResetDBSelection	Database Engine	1.00	No
0x0017	SelectDBRecord	Database Engine As an optimization, selecting a single track or audiobook automatically passes it to the player.	1.00	No
0x0018	GetNumberCategorized-DBRecords	Database Engine	1.00	No
0x0019	ReturnNumber-CategorizedDBRecords	N/A	1.00	No
0x001A	RetrieveCategorized-DatabaseRecords	Database Engine	1.00	No
0x001B	ReturnCategorized-DatabaseRecord	N/A	1.00	No
0x001C	GetPlayStatus	Playback Engine	1.00	No
0x001D	ReturnPlayStatus	N/A	1.00	No
0x001E	GetCurrentPlaying-TrackIndex	Playback Engine	1.00	No
0x001F	ReturnCurrentPlaying-TrackIndex	N/A	1.00	No
0x0020	GetIndexedPlaying-TrackTitle	Playback Engine	1.00	No
0x0021	ReturnIndexedPlaying-TrackTitle	N/A	1.00	No
0x0022	GetIndexedPlaying-TrackArtistName	Playback Engine	1.00	No

Command ID	Command	Target engine	Introduced in protocol version	Requires authentication over serial link
0x0023	ReturnIndexedPlaying-TrackArtistName	N/A	1.00	No
0x0024	GetIndexedPlaying-TrackAlbumName	Playback Engine	1.00	No
0x0025	ReturnIndexedPlaying-TrackAlbumName	N/A	1.00	No
0x0026	SetPlayStatusChange-Notification	Playback Engine	1.00	No
0x0027	PlayStatusChange-Notification	N/A	1.00	No
0x0028	PlayCurrentSelection	Database and Playback Engines. This command copies items from the database to the Playback Engine.	1.00	No
0x0029	PlayControl	Playback Engine	1.00	No
0x002A	GetTrackArtworkTimes	Playback Engine	1.10	No
0x002B	RetTrackArtworkTimes	Playback Engine	1.10	No
0x002C	GetShuffle	N/A	1.00	No
0x002D	ReturnShuffle	N/A	1.00	No
0x002E	SetShuffle	N/A	1.00	No
0x002F	GetRepeat	N/A	1.00	No
0x0030	ReturnRepeat	N/A	1.00	No
0x0031	SetRepeat	N/A	1.00	No
0x0032	SetDisplayImage	N/A	1.01	No
0x0033	GetMonoDisplay-ImageLimits	N/A	1.01	No
0x0034	ReturnMonoDisplay-ImageLimits	N/A	1.01	No
0x0035	GetNumPlayingTracks	Playback Engine	1.01	No
0x0036	ReturnNumPlayingTracks	N/A	1.01	No

Command ID	Command	Target engine	Introduced in protocol version	Requires authentication over serial link
0x0037	SetCurrentPlayingTrack	Playback Engine	1.01	No
0x0038	SelectSortDBRecord	Database Engine As an optimization, selecting a single track or audiobook automatically passes it to the player.	1.01	No
0x0039	GetColorDisplay-ImageLimits	N/A	1.09	No
0x003A	ReturnColorDisplay-ImageLimits	N/A	1.09	No
0x003B	ResetDBSelection-Hierarchy	Database Engine	1.11	Yes
0x003C	GetDBiTunesInfo	Database Engine	1.13	Yes
0x003D	RetDBiTunesInfo	Database Engine	1.13	Yes
0x003E	GetUIDTrackInfo	Database Engine	1.13	Yes
0x003F	RetUIDTrackInfo	Database Engine	1.13	Yes
0x0040	GetDBTrackInfo	Database Engine	1.13	Yes
0x0041	RetDBTrackInfo	Database Engine	1.13	Yes
0x0042	GetPBTrackInfo	Playback Engine	1.13	Yes
0x0043	RetPBTrackInfo	Playback Engine	1.13	Yes
0x0044 – 0xFFFF	Reserved	N/A	N/A	N/A

## Command 0x0001: ACK

Direction: iPod to Device

The iPod sends this command to acknowledge the receipt of a command and return the command status. The command ID field indicates the device command for which the response is being sent. The command status indicates the results of the command (success or failure).

**Table 4-11** ACK command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x06	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x01	Command ID (bits 7:0)
6	0xNN	Command result status. Possible values are: <ul style="list-style-type: none"> <li>■ 0x00 = Success (OK)</li> <li>■ 0x01 = ERROR: Unknown database category</li> <li>■ 0x02 = ERROR: Command failed</li> <li>■ 0x03 = ERROR: Out of resources</li> <li>■ 0x04 = ERROR: Bad parameter</li> <li>■ 0x05 = ERROR: Unknown ID</li> <li>■ 0x06 = Reserved</li> <li>■ 0x07 = Accessory not authenticated</li> <li>■ 0x08 – 0xFF = Reserved</li> </ul>
7	0xNN	The ID of the command being acknowledged (bits 15:8).
8	0xNN	The ID of the command being acknowledged (bits 7:0).
9	0xNN	Packet payload checksum byte

## Command 0x0002: GetCurrentPlayingTrackChapterInfo

---

Direction: Device to iPod

Applies To: Playback Engine

Requests the chapter information of the currently playing track. In response, the iPod sends a "[Command 0x0003: ReturnCurrentPlayingTrackChapterInfo](#)" (page 373) command to the device.



**Note:** The returned track index is valid only when there is a currently playing or paused track.

**Table 4-12** `GetCurrentPlayingTrackChapterInfo` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x02	Command ID (bits 7:0)
6	0xF7	Packet payload checksum byte

## Command 0x0003: `ReturnCurrentPlayingTrackChapterInfo`

Direction: iPod to Device

Returns the chapter information of the currently playing track. The iPod sends this command in response to the "[Command 0x0002: `GetCurrentPlayingTrackChapterInfo`](#)" (page 372) command from the device. The track chapter information includes the currently playing track's chapter index, as well as the total number of chapters in the track. The track chapter and the total number of chapters are 32-bit signed integers. The chapter index of the first chapter is always 0x00000000. If the track does not have chapter information, a chapter index of -1 (0xFFFFFFFF) and a chapter count of 0 (0x00000000) are returned.

**Table 4-13** `ReturnCurrentPlayingTrackChapterInfo` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0B	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x03	Command ID (bits 7:0)
6	0xNN	Current chapter index (bits 31:24)
7	0xNN	Current chapter index (bits 23:16)
8	0xNN	Current chapter index (bits 15:8)

Byte number	Value	Meaning
9	0xNN	Current chapter index (bits 7:0)
10	0xNN	Chapter count (bits 31:24)
11	0xNN	Chapter count (bits 23:16)
12	0xNN	Chapter count (bits 15:8)
13	0xNN	Chapter count (bits 7:0)
14	0xNN	Packet payload checksum byte

## Command 0x0004: SetCurrentPlayingTrackChapter

Direction: Device to iPod

Applies To: Playback Engine

Sets the currently playing track chapter. You can send the ["Command 0x0002: GetCurrentPlayingTrackChapterInfo"](#) (page 372) command to get the chapter count and the index of the currently playing chapter in the current track. In response to the `SetCurrentPlayingTrackChapter` command, the iPod sends an ACK command with the command status.

**Note:** This command should be used only when the iPod is in a playing or paused state. The command fails if the iPod is stopped or if the track does not contain chapter information.

**Table 4-14** `SetCurrentPlayingTrackChapter` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x04	Command ID (bits 7:0)
6	0xNN	Chapter index (bits 31:24)
7	0xNN	Chapter index (bits 23:16)
8	0xNN	Chapter index (bits 15:8)
9	0xNN	Chapter index (bits 7:0)

Byte number	Value	Meaning
10	0xNN	Packet payload checksum byte

## Command 0x0005: GetCurrentPlayingTrackChapterPlayStatus

Direction: Device to iPod

Applies To: Playback Engine

Requests the chapter playtime status of the currently playing track. The status includes the chapter length and the time elapsed within that chapter. In response to a valid command, the iPod sends a "[Command 0x0006: ReturnCurrentPlayingTrackChapterPlayStatus](#)" (page 375) command to the device.

**Note:** If the packet length or chapter index is invalid—for instance, if the track does not contain chapter information—the iPod responds with an ACK command including the specific error status.

**Table 4-15** GetCurrentPlayingTrackChapterPlayStatus command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x05	Command ID (bits 7:0)
6	0xNN	Currently playing chapter index (bits 31:24)
7	0xNN	Currently playing chapter index (bits 23:16)
8	0xNN	Currently playing chapter index (bits 15:8)
9	0xNN	Currently playing chapter index (bits 7:0)
10	0xNN	Packet payload checksum byte

## Command 0x0006: ReturnCurrentPlayingTrackChapterPlayStatus

Direction: iPod to Device

Returns the play status of the currently playing track chapter. The iPod sends this command in response to the ["Command 0x0005: GetCurrentPlayingTrackChapterPlayStatus"](#) (page 375) command from the device. The returned information includes the chapter length and elapsed time, in milliseconds. If there is no currently playing chapter, the chapter length and elapsed time are zero.

**Table 4-16** ReturnCurrentPlayingTrackChapterPlayStatus command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0B	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x06	Command ID (bits 7:0)
6	0xNN	Chapter length in milliseconds (bits 31:24)
7	0xNN	Chapter length in milliseconds (bits 23:16)
8	0xNN	Chapter length in milliseconds (bits 15:8)
9	0xNN	Chapter length in milliseconds (bits 7:0)
10	0xNN	Elapsed time in chapter, in milliseconds (bits 31:24)
11	0xNN	Elapsed time in chapter, in milliseconds (bits 23:16)
12	0xNN	Elapsed time in chapter, in milliseconds (bits 15:8)
13	0xNN	Elapsed time in chapter, in milliseconds (bits 7:0)
14	0xNN	Packet payload checksum byte

## Command 0x0007: GetCurrentPlayingTrackChapterName

---

Direction: Device to iPod

Applies To: Playback Engine

Requests a chapter name in the currently playing track. In response to a valid command, the iPod sends a ["Command 0x0008: ReturnCurrentPlayingTrackChapterName"](#) (page 377) command to the device.

**Note:** If the received packet length or track index is invalid—for instance, if the track does not have chapter information or is not a part of the Audiobook category—the iPod responds with an ACK command including the specific error status.

**Table 4-17** `GetCurrentPlayingTrackChapterName` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x07	Command ID (bits 7:0)
6	0xNN	Chapter index (bits 31:24)
7	0xNN	Chapter index (bits 23:16)
8	0xNN	Chapter index (bits 15:8)
9	0xNN	Chapter index (bits 7:0)
10	0xNN	Packet payload checksum byte

## Command 0x0008: `ReturnCurrentPlayingTrackChapterName`

Direction: iPod to Device

Returns a chapter name in the currently playing track. The iPod sends this command in response to a valid "[Command 0x0007: `GetCurrentPlayingTrackChapterName`](#)" (page 376) command from the device. The chapter name is encoded as a null-terminated UTF-8 character array.

**Note:** The chapter name string is not limited to 252 characters; it may be sent in small or large packet format depending on the string length. The small packet format is shown.

**Table 4-18** `ReturnCurrentPlayingTrackChapterName` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length

Byte number	Value	Meaning
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x08	Command ID (bits 7:0)
6... <i>N</i>	0x <i>NN</i>	Chapter name as UTF-8 character array
(last byte)	0x <i>NN</i>	Packet payload checksum byte

## Command 0x0009: GetAudiobookSpeed

---

Direction: Device to iPod

Requests the current iPod audiobook speed state. The iPod responds with the "[Command 0x000A: ReturnAudiobookSpeed](#)" (page 378) command indicating the current audiobook speed.

**Table 4-19** GetAudiobookSpeed command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x09	Command ID (bits 7:0)
6	0xF0	Packet payload checksum byte

## Command 0x000A: ReturnAudiobookSpeed

---

Direction: iPod to Device

Returns the current audiobook speed setting. The iPod sends this command in response to the "[Command 0x0009: GetAudiobookSpeed](#)" (page 378) command from the device.

**Table 4-20** ReturnAudiobookSpeed command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0A	Command ID (bits 7:0)
6	0xNN	Audiobook speed status code. See <a href="#">Table 4-21</a> (page 379).
7	0xNN	Packet payload checksum byte

[Table 4-21](#) (page 379) shows the possible audiobook speed states returned by this command.

**Table 4-21** Audiobook speed states

Value	Meaning
0xFF	Slower (–1)
0x00	Normal
0x01	Faster (+1)
0x02 – 0xFE	Reserved

## Command 0x000B: SetAudiobookSpeed

Direction: Device to iPod

Sets the speed of audiobook playback. The iPod audiobook speed states are listed in [Table 4-21](#) (page 379). This command has two modes: one to set the speed of the currently playing audiobook and a second to set the audiobook speed for all audiobooks.

Byte number 7 is an optional byte; devices must not send this byte if they want to set the speed only of the currently playing audiobook. If devices want to set the global audiobook speed setting then they must use byte number 7. This is the Restore on Exit byte; a nonzero value restores the original audiobook speed setting when the accessory is detached. If this byte is zero, the audiobook speed setting set by the accessory is saved and persists after the accessory is detached from the iPod. See [Table 4-23](#) (page 380) for the packet format used when including the Restore on Exit byte.

In response to this command, the iPod sends an ACK command with the command status.

**Note:** Accessory developers are encouraged to always use the Restore on Exit byte with a nonzero value, to restore any of the user's iPod settings that were modified by the accessory.

**Table 4-22** SetAudiobookSpeed command to set the speed of the currently playing audiobook

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0B	Command ID (bits 7:0)
6	0xNN	New audiobook speed code. See <a href="#">Table 4-21</a> (page 379).
7	0xNN	Packet payload checksum byte

[Table 4-23](#) (page 380) shows a command to set the global audiobook speed for all audiobooks and optionally restore the setting on accessory detach.

**Table 4-23** SetAudiobookSpeed command to set the global audiobook speed

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x05	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0B	Command ID (bits 7:0)
6	0xNN	Global audiobook speed code. See <a href="#">Table 4-21</a> (page 379).
7	0xNN	Restore on Exit byte. If 1, the original setting is restored on detach; if 0, the new setting persists after accessory detach.
8	0xNN	Packet payload checksum byte

## Command 0x000C: GetIndexedPlayingTrackInfo

Direction: Device to iPod



## Applies To: Playback Engine

Gets track information for the track at the specified index. The track info type field specifies the type of information to be returned, such as current song lyrics, podcast name, episode date, and episode description. In response, the iPod sends the "[Command 0x000D: ReturnIndexedPlayingTrackInfo](#)" (page 382) command with the requested track information. If the information type is invalid or does not apply to the selected track, the iPod returns an ACK with an error status.

A device can determine the number of tracks in the list of tracks queued to play on the iPod by sending a "[GetNumPlayingTracks](#)" (page 423) command and receiving a "[ReturnNumPlayingTracks](#)" (page 424) command in reply.

**Table 4-24** GetIndexedPlayingTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0A	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0C	Command ID (bits 7:0)
6	0xNN	Track info type. See <a href="#">Table 4-25</a> (page 381).
7	0xNN	Track index (bits 31:24)
8	0xNN	Track index (bits 23:16)
9	0xNN	Track index (bits 15:8)
10	0xNN	Track index (bits 7:0)
11	0xNN	Chapter index (bits 15:8)
12	0xNN	Chapter index (bits 7:0)
(last byte)	0xNN	Packet payload checksum byte

[Table 4-25](#) (page 381) shows the types of information you can obtain for a track.

**Table 4-25** Track information type

Info Type	Code
0x00	Track Capabilities and Information
0x01	Podcast Name
0x02	Track Release Date

Info Type	Code
0x03	Track Description
0x04	Track Song Lyrics
0x05	Track Genre
0x06	Track Composer
0x07	Track Artwork Count

**Note:** Requesting the lyrics of a track not currently being played will result in a null string being returned. Lyrics for a track being played may take up to 5 seconds to return. Use `SetPlayStatusChangeNotification` to be notified when the lyrics for the currently playing track become available.

## Command 0x000D: ReturnIndexedPlayingTrackInfo

Direction: iPod to Device

Returns the requested track information type and data. The iPod sends this command in response to the "Command 0x000C: GetIndexedPlayingTrackInfo" (page 380) command. Table 4-28 (page 384) lists the available information types and their data.

Data returned as strings are encoded as null-terminated UTF-8 character arrays. If the track information string does not exist, a null UTF-8 string is returned. If the track has no release date, then the returned release date has all bytes zeros. Track song lyrics and the track description are sent in a large or small packet format with an incrementing packet index field, spanning multiple packets if needed.

Table 4-26 (page 382) shows the packet format for the `ReturnIndexedPlayingTrackInfo` command sent in response to a request for information types 0x00 to 0x02.

**Table 4-26** `ReturnIndexedPlayingTrackInfo` command for info types 0x00-0x02 and 0x05-0x07

Byte number	Value	Meaning
0	0xFF	Sync byte
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0D	Command ID (bits 7:0)
6	0xNN	Track info type. See Table 4-28 (page 384).

Byte number	Value	Meaning
7 ... <i>N</i>	0x <i>NN</i>	Track information. The data format is specific to the track info type. See <a href="#">Table 4-28</a> (page 384).
(last byte)	0x <i>NN</i>	Packet payload checksum byte

[Table 4-27](#) (page 383) shows the packet format for the `ReturnIndexedPlayingTrackInfo` command sent in response to a request for information types 0x03 to 0x04. The small packet format is not shown.

**Table 4-27** `ReturnIndexedPlayingTrackInfo` command for info types 0x03-0x04

Byte number	Value	Meaning
0	0xFF	Sync byte
1	0x55	Start of packet
2	0x00	Packet payload marker (large format)
3	0x <i>NN</i>	Large packet payload length (bits 15:8)
4	0x <i>NN</i>	Large packet payload length (bits 7:0)
5	0x04	Lingo ID (Extended Interface lingo)
6	0x00	Command ID (bits 15:8)
7	0x0D	Command ID (bits 7:0)
8	0x <i>NN</i>	Track info type. See <a href="#">Table 4-28</a> (page 384).
9	0x <i>NN</i>	Packet information bits. If set, these bits have the following meanings: <ul style="list-style-type: none"> <li>■ Bit 31:2 Reserved</li> <li>■ Bit 1: This is the last packet. Applicable only if bit 0 is set.</li> <li>■ Bit 0: Indicates that there are multiple packets.</li> </ul>
10	0x <i>NN</i>	Packet Index (bits 15:8)
11	0x <i>NN</i>	Packet Index (bits 7:0)
12 ... <i>N</i>	0x <i>NN</i>	Track information as a UTF-8 string.
(last byte)	0x <i>NN</i>	Packet payload checksum byte

[Table 4-28](#) (page 384) shows the available track information types and the format of the data returned for each type.

**Table 4-28** Track info types and return data

Info Type	Code	Data Format
0x00	Track Capabilities and Information	A 10-byte data. See <a href="#">Table 4-29</a> (page 384).
0x01	Podcast Name	UTF-8 string
0x02	Track Release Date	An 8-byte data. See <a href="#">Table 4-30</a> (page 385)
0x03	Track Description	UTF-8 string
0x04	Track Song Lyrics	UTF-8 string
0x05	Track Genre	UTF-8 string
0x06	Track Composer	UTF-8 string
0x07	Track Artwork Count	Artwork count data. The artwork count is a sequence of 4-byte records; each record consists of a 2-byte format ID value followed by a 2-byte count of images in that format for this track. For more information about formatID and chapter index values, see commands 0x000E-0x0011 and 0x002A-0x002B.
0x08-0xFF	Reserved	N/A

[Table 4-29](#) (page 384) shows the data returned for the Track Capabilities and Information type.

**Table 4-29** Track Capabilities and Information encoding

Byte number	Code
0x00-0x03	Track Capability bits. If set, these bits have the following meanings: <ul style="list-style-type: none"> <li>■ Bit 31:9: Reserved</li> <li>■ Bit 8: Track is currently queued to play as a video</li> <li>■ Bit 7: Track contains video (a video podcast, music video, movie, or TV show)</li> <li>■ Bit 6: Track has description</li> <li>■ Bit 5: Track has release date</li> <li>■ Bit 4: Track is a podcast episode</li> <li>■ Bit 3: Track has song lyrics</li> <li>■ Bit 2: Track has album artwork</li> <li>■ Bit 1: Track has chapters</li> <li>■ Bit 0: Track is audiobook</li> </ul>
0x04	Total track length, in milliseconds (bits 31:24)
0x05	Total track length, in milliseconds (bits 23:16)

Byte number	Code
0x06	Total track length, in milliseconds (bits 15:8)
0x07	Total track length, in milliseconds (bits 7:0)
0x08	Chapter count (bits 15:8)
0x09	Chapter count (bits 7:0)

Table 4-30 (page 385) shows the data returned for the Track Release Date type.

**Table 4-30** Track Release Date encoding

Byte number	Code
0x00	Seconds (0-59)
0x01	Minutes (0-59)
0x02	Hours (0-23)
0x03	Day of the month (1-31)
0x04	Month (1-12)
0x05	Year (bits 15:8). For example, 2006 signifies the year 2006 AD.
0x06	Year (bits 7:0).
0x07	Weekday (0-6, where 0 = Sunday and 6 = Saturday)

## Command 0x000E: GetArtworkFormats

---

Direction: Device to iPod

Applies To: Playback Engine

The device sends this command to obtain the list of supported artwork formats on the iPod. No parameters are sent. See ["Transferring Album Art"](#) (page 354).

**Table 4-31** GetArtworkFormats packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Length of packet
3	0x04	Lingo ID: Extended Interface lingo

Byte number	Value	Comment
4	0x00	Command ID (bits 15:8)
5	0x0E	Command ID (bits 7:0)
6	0xEB	Checksum

## Command 0x000F: RetArtworkFormats

Direction: iPod to Device

Applies To: Playback Engine

The iPod sends this command to the device, giving it the list of supported artwork formats. Each format is described in a 7-byte record (formatID:2, pixelFormat:1, width:2, height:2). The `formatID` is used when sending `GetTrackArtworkTimes`. The device may return zero records. See ["Transferring Album Art"](#) (page 354).

**Table 4-32** RetArtworkFormats packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x0F	Command ID (bits 7:0)
NN	0xNN	formatID (15:8) iPod-assigned value for this format
NN	0xNN	formatID (7:0)
NN	0xNN	pixelFormat. Same as from <code>SetDisplayImage</code>
NN	0xNN	imageWidth (15:8). Number of pixels wide for each image.
NN	0xNN	imageWidth (7:0)
NN	0xNN	imageHeight (15:8). Number of pixels high for each image.
NN	0xNN	imageHeight (7:0)
Previous 7 bytes may be repeated NN times		
(last byte)	0xNN	Checksum

## Command 0x0010: GetTrackArtworkData

---

Direction: Device to iPod

Applies To: Playback Engine

The device sends this command to the iPod to request data for a given `trackIndex`, `formatID`, and `artworkIndex`. See ["Transferring Album Art"](#) (page 354). The time offset from track start is the value returned by ["GetTrackArtworkTimes"](#) (page 409).

**Table 4-33** GetTrackArtworkData packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0D	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x10	Command ID (bits 7:0)
6	0xNN	trackIndex (31:24).
7	0xNN	trackIndex (23:16)
8	0xNN	trackIndex (15:8)
9	0xNN	trackIndex (7:0)
10	0xNN	formatID (15:8)
11	0xNN	formatID (7:0)
12	0xNN	time offset from track start, in ms (31:24)
13	0xNN	time offset from track start, in ms (23:16)
14	0xNN	time offset from track start, in ms (15:8)
15	0xNN	time offset from track start, in ms (7:0)
16	0xNN	Checksum

## Command 0x0011: RetTrackArtworkData

---

Direction: iPod to Device

Applies To: Playback Engine

The iPod sends the requested artwork to the accessory. Multiple `RetTrackArtworkData` commands may be necessary to transfer all the data because it will be too much to fit into a single packet. See ["Transferring Album Art"](#) (page 354).

This command uses nearly the same format as the `SetDisplayImage` command (command 0x0032). The only difference is the addition of 2 coordinates; they define an inset rectangle that describes any padding that may have been added to the image. The coordinates consist of two x,y pairs. Each x or y value is 2 bytes, so the total size of the coordinate set is 8 bytes.

**Table 4-34** `RetTrackArtworkData` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x11	Command ID (bits 7:0)
6	0x00	Descriptor packet index (15:8). These fields uniquely identify each packet in the <code>RetTrackArtworkData</code> transaction. The first command is the descriptor packet, which always starts with an index of 0x0000.
7	0x00	Descriptor packet index (7:0)
8	0xNN	Display pixel format code.
9	0xNN	Image width in pixels (15:8)
10	0xNN	Image width in pixels (7:0)
11	0xNN	Image height in pixels (15:8)
12	0xNN	Image height in pixels (7:0)
13	0xNN	Inset rectangle, top-left point, x value (15:8)
14	0xNN	Inset rectangle, top-left point, x value (7:0)
15	0xNN	Inset rectangle, top-left point, y value (15:8)
16	0xNN	Inset rectangle, top-left point, y value (7:0)
17	0xNN	Inset rectangle, bottom-right point, x value (15:8)
18	0xNN	Inset rectangle, bottom-right point, x value (7:0)
19	0xNN	Inset rectangle, bottom-right point, y value (15:8)
20	0xNN	Inset rectangle, bottom-right point, y value (7:0)



Byte number	Value	Comment
21	0xNN	Row size in bytes (31:24)
22	0xNN	Row size in bytes (23:16)
23	0xNN	Row size in bytes (15:8)
24	0xNN	Row size in bytes (7:0)
25-NN	0xNN...	Image pixel data (variable length)
(last byte)	0xNN	Checksum

In subsequent packets in the sequence, bytes 8 through 24 are omitted.

## Command 0x0016: ResetDBSelection

Direction: Device to iPod

Applies To: Database Engine

Resets the current database selection to an empty state, invalidates the category entry count (sets the count to 0) for all categories except the playlist category, and sets the database hierarchy to the audio hierarchy (even if it is currently in the video hierarchy). This is analogous to pressing the Menu button repeatedly to get to the topmost iPod HMI menu. Any previously selected database items are deselected. The command has no effect on the Playback Engine. In response, the iPod sends an ACK command with the command status.

Once the accessory has reset the database selection, it must initialize the category count before it can select database records. Please refer to ["Command 0x0018: GetNumberCategorizedDBRecords"](#) (page 392) and ["Command 0x0017: SelectDBRecord"](#) (page 390) for details.

**Note:** Starting with protocol version 1.07, the `ResetDBSelection` command clears the sort order.

**Table 4-35** `ResetDBSelection` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x16	Command ID (bits 7:0)

Byte number	Value	Meaning
6	0xE3	Packet payload checksum byte

## Command 0x0017: SelectDBRecord

Direction: Device to iPod

Applies To: Database Engine. Selecting a single track automatically passes it to the Playback Engine.

Selects one or more records in the Database Engine, based on a category relative index. For example, selecting category two (artist) and record index one results in a list of selected tracks (or database records) from the second artist in the artist list. [Table 4-37](#) (page 391) lists the available database categories.

**Note:** With the iPhone and iPod touch, selecting a specific track in a playlist containing only skip-when-shuffle tracks causes the entire playlist to be shuffled and passed to the Playback Engine. With other iPods, passing a valid track index with `SelectDBRecord` and a playlist containing only skip-when-shuffle tracks causes a single track to be passed to the Playback Engine. Use the `GetNumPlayingTracks` command to query the number of songs in the Playback Engine before using any other Playback Engine commands to alter playback.

Selections are additive and limited by the category hierarchy; see ["Database Category Hierarchies"](#) (page 350) for more information about category hierarchies. Subsequent selections are made based on the subset of records resulting from the previous selections and not from the entire database. Note that the selection of a single record automatically passes it to the Playback Engine and starts its playback. Record indices consist of a 32-bit signed integer. To select database records with a specific sort order, use ["Command 0x0038: SelectSortDBRecord"](#) (page 425).

`SelectDBRecord` may be called only after a category count has been initialized through a call to ["Command 0x0018: GetNumberCategorizedDBRecords"](#) (page 392). Without a valid category count, the `SelectDBRecord` call cannot select a database record and the result of calling it will be undefined. Accessories that make use of ["Command 0x0016: ResetDBSelection"](#) (page 389) must always initialize the category count before selecting a new database record using `SelectDBRecord`.

Accessories should pay close attention to the ACK returned by the `SelectDBRecord` command. Ignoring errors may cause unexpected behavior.

To undo a database selection, send the `SelectDBRecord` command with the current category selected in the Database Engine and a record index of `-1` (0xFFFFFFFF). This has the same effect as pressing the iPod Menu button once and moves the database selection up to the next highest menu level. For example, if a device selected artist number three and then album number one, it could use the `SelectDBRecord(Album, -1)` command to return to the database selection of artist number three. If multiple database selections have been made, devices can use any of the previously used categories to return to the next highest database selection. If the category used in one of these `SelectDBRecord` commands has not been used in a previous database selection then the command is treated as a no-op.

Sending a `SelectDBRecord` command with the Track or Audiobook category and a record index of `-1` is invalid, because the previous database selection made with the Track category and a valid index passes the database selection to the Playback Engine. Sending a `SelectDBRecord(Track, -1)` command returns a parameter error. The iPod also returns a bad parameter error ACK when devices send the `SelectDBRecord` command with an invalid category type, or with the Track category and an index greater than the total number of tracks available on the iPod.

**Note:** Selecting a podcast always selects from the main podcast library regardless of the current category context of the iPod.

To immediately go to the topmost iPod menu level and reset all database selections, send the `ResetDBSelection` command to the iPod.

**Table 4-36** `SelectDBRecord` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x08	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x17	Command ID (bits 7:0)
6	0xNN	Database category type. See <a href="#">Table 4-37</a> (page 391).
7	0xNN	Database record index (bits 31:24)
8	0xNN	Database record index (bits 23:16)
9	0xNN	Database record index (bits 15:8)
10	0xNN	Database record index (bits 7:0)
11	0xNN	Packet payload checksum byte

[Table 4-37](#) (page 391) lists the valid database categories.

**Table 4-37** Database category types for commands

Category	Code	Protocol version
Reserved	0x00	N/A
Playlist	0x01	1.00
Artist	0x02	1.00
Album	0x03	1.00
Genre	0x04	1.00
Track	0x05	1.00
Composer	0x06	1.00

Category	Code	Protocol version
Audiobook	0x07	1.06
Podcast	0x08	1.08
Nested playlist	0x09	1.13
Reserved	0x0A – 0xFF	N/A

## Command 0x0018: GetNumberCategorizedDBRecords

Direction: Device to iPod

Applies To: Database Engine

Retrieves the number of records in a particular database category. For example, a device can get the number of artists or albums present in the database. The category types are described in [Table 4-37](#) (page 391). The iPod responds with a ["Command 0x0019: ReturnNumberCategorizedDBRecords"](#) (page 393) command indicating the number of records present for this category.

`GetNumberCategorizedDBRecords` must be called to initialize the category count before selecting a database record using ["Command 0x0017: SelectDBRecord"](#) (page 390) or ["Command 0x0038: SelectSortDBRecord"](#) (page 425) commands. A category's record count can change based on the prior categories selected and the database hierarchy. The accessory is expected to call `GetNumberCategorizedDBRecords` in order to get the valid range of category entries before selecting a record in that category.

**Note:** The record count returned by this command depends on the database state before this command is sent. If the database has been reset using ["Command 0x0016: ResetDBSelection"](#) (page 389), this command returns the total number of records for a given category. However, if this command is sent after one or more categories are selected, the record count is the subset of records that are members of all the categories selected prior to this command.

**Table 4-38** `GetNumberCategorizedDBRecords` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x18	Command ID (bits 7:0)
6	0xNN	Database category type. See <a href="#">Table 4-37</a> (page 391).

Byte number	Value	Meaning
7	0xNN	Packet payload checksum byte

## Command 0x0019: ReturnNumberCategorizedDBRecords

---

Direction: iPod to Device

Returns the number of database records matching the specified database category. The iPod sends this command in response to the ["Command 0x0018: GetNumberCategorizedDBRecords"](#) (page 392) command from the device. Individual records can then be extracted by sending ["Command 0x001A: RetrieveCategorizedDatabaseRecords"](#) (page 393) to the iPod.

If no matching database records are found, a record count of zero is returned. Category types are described in [Table 4-37](#) (page 391).

After selecting the podcast category, the number of artist, album, composer, genre, and audiobook records is always zero.

**Table 4-39** ReturnNumberCategorizedDBRecords command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x19	Command ID (bits 7:0)
6	0xNN	Database record count (bits 31:24)
7	0xNN	Database record count (bits 23:16)
8	0xNN	Database record count (bits 15:8)
9	0xNN	Database record count (bits 7:0)
10	0xNN	Packet payload checksum byte

## Command 0x001A: RetrieveCategorizedDatabaseRecords

---

Direction: Device to iPod

Applies To: Database Engine

Retrieves one or more database records from the iPod, typically based on the results from the "[Command 0x0018: GetNumberCategorizedDBRecords](#)" (page 392) query. The database category types are described in [Table 4-37](#) (page 391).

This command specifies the starting record index and the number of records to retrieve (the record count). This allows a device to retrieve an individual record or the entire set of records for a category. The record start index and record count consist of 32-bit signed integers. To retrieve all records from a given starting record index, set the record count to -1 (0xFFFFFFFF).

The iPod responds to this command with a separate "[Command 0x001B: ReturnCategorizedDatabaseRecord](#)" (page 394) command for each record matching the specified criteria (category and record index range).

**Table 4-40** RetrieveCategorizedDatabaseRecords command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0C	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1A	Command ID (bits 7:0)
6	0xNN	Database category type. See <a href="#">Table 4-37</a> (page 391).
7	0xNN	Database record start index (bits 31:24)
8	0xNN	Database record start index (bits 23:16)
9	0xNN	Database record start index (bits 15:8)
10	0xNN	Database record start index (bits 7:0)
11	0xNN	Database record read count (bits 31:24)
12	0xNN	Database record read count (bits 23:16)
13	0xNN	Database record read count (bits 15:8)
14	0xNN	Database record read count (bits 7:0)
15	0xNN	Packet payload checksum byte

## Command 0x001B: ReturnCategorizedDatabaseRecord

Direction: iPod to Device

Contains information for a single database record. The iPod sends one or more of these commands in response to the "[Command 0x001A: RetrieveCategorizedDatabaseRecords](#)" (page 393) command from the device. The category record index is included to allow the device to determine which record has been sent. The record data is sent as a null-terminated UTF-8 encoded data array.

**Note:** The database record string is not limited to 252 characters; it may be sent in small or large packet format, depending on the record size. The small packet format is shown.

**Table 4-41** ReturnCategorizedDatabaseRecord command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1B	Command ID (bits 7:0)
6	0xNN	Database record category index (bits 31:24)
7	0xNN	Database record category index (bits 23:16)
8	0xNN	Database record category index (bits 15:8)
9	0xNN	Database record category index (bits 7:0)
10...N	0xNN	Database record as a UTF-8 character array.
(last byte)	0xNN	Packet payload checksum byte

## Command 0x001C: GetPlayStatus

Direction: Device to iPod

Applies To: Playback Engine

Requests the current iPod playback status, allowing the device to display feedback to the user. In response, the iPod sends a "[Command 0x001D: ReturnPlayStatus](#)" (page 396) command with the current playback status.

**Note:** An accessory must not call `GetPlayStatus` more often than once a second. Instead of polling for the play status with this command, the accessory should request notifications from the iPod using `SetPlayStatusChangeNotification`.

**Table 4-42** `GetPlayStatus` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1C	Command ID (bits 7:0)
6	0xDD	Packet payload checksum byte

## Command 0x001D: `ReturnPlayStatus`

Direction: iPod to Device

Returns the current iPod playback status. The iPod sends this command in response to the "[Command 0x001C: `GetPlayStatus`](#)" (page 395) command from the device. The information returned includes the current track length, track position, and player state.

**Note:** The track length and track position fields are valid only if the player state is Playing or Paused. For other player states, these fields must be ignored.

**Table 4-43** `ReturnPlayStatus` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x0C	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1D	Command ID (bits 7:0)
6	0xNN	Track length in milliseconds (bits 31:24)



Byte number	Value	Meaning
7	0xNN	Track length in milliseconds (bits 23:16)
8	0xNN	Track length in milliseconds (bits 15:8)
9	0xNN	Track length in milliseconds (bits 7:0)
10	0xNN	Track position in milliseconds (bits 31:24)
11	0xNN	Track position in milliseconds (bits 23:16)
12	0xNN	Track position in milliseconds (bits 15:8)
13	0xNN	Track position in milliseconds (bits 7:0)
14	0xNN	Player state. Possible values are: <ul style="list-style-type: none"> <li>■ 0x00 = Stopped</li> <li>■ 0x01 = Playing</li> <li>■ 0x02 = Paused</li> <li>■ 0x03 – 0xFE = Reserved</li> <li>■ 0xFF = Error</li> </ul>
15	0xNN	Packet payload checksum byte

## Command 0x001E: GetCurrentPlayingTrackIndex

Direction: Device to iPod

Applies To: Playback Engine

Requests the Playback Engine index of the currently playing track. In response, the iPod sends a ["Command 0x001F: ReturnCurrentPlayingTrackIndex"](#) (page 398) command to the device.

**Note:** The track index returned is valid only if there is currently a track playing or paused.

**Table 4-44** GetCurrentPlayingTrackIndex command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)

Byte number	Value	Meaning
5	0x1E	Command ID (bits 7:0)
6	0xDB	Packet payload checksum byte

## Command 0x001F: ReturnCurrentPlayingTrackIndex

---

Direction: iPod to Device

Returns the Playback Engine index of the current playing track in response to the "[Command 0x001E: GetCurrentPlayingTrackIndex](#)" (page 397) command from the device. The track index is a 32-bit signed integer. If there is no track currently playing or paused, an index of -1 (0xFFFFFFFF) is returned.

**Table 4-45** ReturnCurrentPlayingTrackIndex command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x1F	Command ID (bits 7:0)
6	0xNN	Playback track index (bits 31:24)
7	0xNN	Playback track index (bits 23:16)
8	0xNN	Playback track index (bits 15:8)
9	0xNN	Playback track index (bits 7:0)
10	0xNN	Packet payload checksum byte

## Command 0x0020: GetIndexedPlayingTrackTitle

---

Direction: Device to iPod

Applies To: Playback Engine

Requests the title name of the indexed playing track from the iPod. In response to a valid command, the iPod sends a "[Command 0x0021: ReturnIndexedPlayingTrackTitle](#)" (page 399) command to the device.

**Note:** If the packet length or playing track index is invalid, the iPod responds with an ACK command including the specific error status.

**Table 4-46** GetIndexedPlayingTrackTitle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x20	Command ID (bits 7:0)
6	0xNN	Playback track index (bits 31:24)
7	0xNN	Playback track index (bits 23:16)
8	0xNN	Playback track index (bits 15:8)
9	0xNN	Playback track index (bits 7:0)
10	0xNN	Packet payload checksum byte

## Command 0x0021: ReturnIndexedPlayingTrackTitle

Direction: iPod to Device

Returns the title of the indexed playing track in response to a valid "Command 0x0020: GetIndexedPlayingTrackTitle" (page 398) command from the device. The track title is encoded as a null-terminated UTF-8 character array.

**Note:** The track title string is not limited to 252 characters; it may be sent in small or large packet format, depending on the string length. The small packet format is shown.

**Table 4-47** ReturnIndexedPlayingTrackTitle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo

Byte number	Value	Meaning
4	0x00	Command ID (bits 15:8)
5	0x21	Command ID (bits 7:0)
6... <i>N</i>	0x <i>NN</i>	Track title as a UTF-8 character array
(last byte)	0x <i>NN</i>	Packet payload checksum byte

## Command 0x0022: GetIndexedPlayingTrackArtistName

Direction: Device to iPod

Applies To: Playback Engine

Requests the name of the artist of the indexed playing track. In response to a valid command, the iPod sends a ["Command 0x0023: ReturnIndexedPlayingTrackArtistName"](#) (page 401) command to the device.

**Note:** If the packet length or playing track index is invalid, the iPod responds with an ACK command including the specific error status.

**Table 4-48** GetIndexedPlayingTrackArtistName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x22	Command ID (bits 7:0)
6	0x <i>NN</i>	Playback track index (bits 31:24)
7	0x <i>NN</i>	Playback track index (bits 23:16)
8	0x <i>NN</i>	Playback track index (bits 15:8)
9	0x <i>NN</i>	Playback track index (bits 7:0)
10	0x <i>NN</i>	Packet payload checksum byte

## Command 0x0023: ReturnIndexedPlayingTrackArtistName

Direction: iPod to Device

Returns the artist name of the indexed playing track in response to a valid "Command 0x0022: GetIndexedPlayingTrackArtistName" (page 400) command from the device. The track artist name is encoded as a null-terminated UTF-8 character array.

**Note:** The artist name string is not limited to 252 characters; it may be sent in small or large packet format, depending on the string length. The small packet format is shown.

**Table 4-49** ReturnIndexedPlayingTrackArtistName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x23	Command ID (bits 7:0)
6...N	0xNN	Artist name as UTF-8 character array
(last byte)	0xNN	Packet payload checksum byte

## Command 0x0024: GetIndexedPlayingTrackAlbumName

Direction: Device to iPod

Applies To: Playback Engine

Requests the album name of the indexed playing track. In response to a valid command, the iPod sends a "Command 0x0025: ReturnIndexedPlayingTrackAlbumName" (page 402) command to the device.

**Note:** If the received packet length or playing track index is invalid, the iPod responds with an ACK command including the specific error status.

**Table 4-50** GetIndexedPlayingTrackAlbumName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x24	Command ID (bits 7:0)
6	0xNN	Playback track index (bits 31:24)
7	0xNN	Playback track index (bits 23:16)
8	0xNN	Playback track index (bits 15:8)
9	0xNN	Playback track index (bits 7:0)
10	0xNN	Packet payload checksum byte

## Command 0x0025: ReturnIndexedPlayingTrackAlbumName

Direction: iPod to Device

Returns the album name of the indexed playing track in response to a valid ["Command 0x0024: GetIndexedPlayingTrackAlbumName"](#) (page 401) command from the device. The track album name is encoded as a null-terminated UTF-8 character array.

**Note:** The album name string is not limited to 252 characters; it may be sent in small or large packet format, depending on the string length. The small packet format is shown.

**Table 4-51** ReturnIndexedPlayingTrackAlbumName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x25	Command ID (bits 7:0)
6...N	0xNN	Album name as a UTF-8 character array
(last byte)	0xNN	Packet payload checksum byte

## Command 0x0026: SetPlayStatusChangeNotification

Direction: Device to iPod

Applies To: Playback Engine

This command is sent by the device to control the status change event types sent by the iPod.

There are two forms of the command. The one-byte form accepts a single Boolean to enable or disable all notifications for play state, track index, track time position, FFW/REW seek stop, and chapter index changes (`StatusChangeNotification` types 0x00-0x05). The packet for this form of the command is shown in [Table 4-52](#) (page 403). When the value of byte 6 is set to 0x00, all notifications are disabled, whether set by the one-byte or four-byte form of the command.

The four-byte form expands the status notification control into individual bits for each type of status. The packet for this form of the command is shown in [Table 4-53](#) (page 403). Bytes 6 through 9 are a fixed-length bitmask of events to be enabled or disabled, where 1 = enable, 0 = disable.

If any notifications are enabled, their notification conditions are tested approximately every 500 milliseconds. When the conditions change, notifications are sent to the device.

**Note:** Status change notifications are not sent when the iPod leaves the power-on state. This means that no notifications are sent while the iPod remains in either the sleep or hibernate states. When the iPod returns to the power-on state, status notifications resume with their previous settings.

**Table 4-52** One-byte `SetPlayStatusChangeNotification` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x26	Command ID (bits 7:0)
6	0xNN	Enable/disable notifications; see <a href="#">Table 4-54</a> (page 404).
7	0xNN	Packet payload checksum byte

**Table 4-53** Four-byte `SetPlayStatusChangeNotification` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x26	Command ID (bits 7:0)
6	0x00	Notification event mask (bits 31:24)
7	0x00	Notification event mask (bits 23:16)
8	0xNN	Notification event mask (bits 15:8); see <a href="#">Table 4-55</a> (page 404)
9	0xNN	Notification event mask (bits 7:0); see <a href="#">Table 4-55</a> (page 404)
10	0xNN	Packet payload checksum byte

**Table 4-54** One-byte status change event values

Value	Description
0x00	Disable all status event notifications
0x01	Enable play status notifications for basic play state, track index, track time position, FFW/REW seek stop, and chapter index changes ( <code>StatusChangeNotification</code> types 0x00-0x05).
0x02-0xFF	Reserved

**Table 4-55** Four-byte status change event mask bits

Bit	Description
00	Basic play state changes (stop, FFW seek stop, or REW seek stop, using status notification types 0x00, 0x02, or 0x03).
01	Extended play state changes (playback stop, FFW seek start, REW seek start, playback started, FFW/REW seek stop, or playback pause using status notification type 0x06). Uses <code>PlayControl</code> command control codes as the play status codes; see <a href="#">Table 4-60</a> (page 408).
02	Track index
03	Track time offset (ms)
04	Track time offset (sec)
05	Chapter index
06	Chapter time offset (ms)
07	Chapter time offset (sec)



Bit	Description
08	Track unique identifier
09	Track media type (audio/video)
10	Track lyrics ready (if the track has lyrics)
11-31	Reserved

## Command 0x0027: PlayStatusChangeNotification

Direction: iPod to Device

This command is sent by the iPod to notify the device about extended interface status changes. The types of status notifications sent to the device are controlled by the ["SetPlayStatusChangeNotification"](#) (page 403) command.

**Note:** Status change notifications are not sent when the iPod leaves the power-on state. This means that no notifications are sent while the iPod remains in either the sleep or hibernate states. When the iPod returns to the power-on state, status notifications resume with their previous settings.

**Table 4-56** PlayStatusChangeNotification command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x27	Command ID (bits 7:0)
6-0xNN	0xNN	New play status. See <a href="#">Table 4-57</a> (page 405).
(last byte)	0xNN	Packet payload checksum byte

**Table 4-57** Play status change notification codes

Status change	Parameters	Description
Playback stopped	{0x00}	
Track index	{0x01, trackIndex; 4}	trackIndex = playback engine track index
Playback FFW seek stop	{0x02}	

Status change	Parameters	Description
Playback REW seek stop	{0x03}	
Track time offset	{0x04, trackOffsetMs; 4}	trackOffsetMs = track time offset in milliseconds
Chapter index	{0x05, chapIndex; 4}	chapIndex = chapter index
Playback status extended	{0x06, playState; 1}	playState = playback state: 0x00-0x01 = Reserved 0x02 = Stopped 0x03-0x04 = Reserved 0x05 = FFW seek started 0x06 = REW seek started 0x07 = FFW/REW seek stopped 0x08-0x09 = Reserved 0x0A = Playing 0x0B = Paused 0x0C-0xFF = Reserved
Track time offset	{0x07, trackOffsetSec; 4}	trackOffsetSec = track time offset in seconds
Chapter time offset (ms)	{0x08, chapTimeMs; 4}	chapTimeMs = chapter time offset in milliseconds
Chapter time offset (sec)	{0x09, chapTimeSec; 4}	chapTimeSec = chapter time offset in seconds
Track unique identifier	{0x0A, trackUID; 8}	trackUID = track unique identifier
Track playback mode	{0x0B, playMode; 1}	playMode = playback mode of current playing track (see Note, below): 0x00 = Audio track 0x01 = Video track 0x02-0xFF = Reserved
Track lyrics ready	{0x0C}	Lyrics for the currently playing track are available for download
Reserved	{0x0D-0xFF}	

**Note:** The playback mode when a track is playing may depend on the database hierarchy that was current when the track was selected. Hybrid video/audio tracks (such as music videos or video podcasts) are queued as audio tracks if selected in the audio DB hierarchy, or queued as video tracks if selected in the video DB hierarchy.

## Command 0x0028: PlayCurrentSelection

Direction: Device to iPod

Applies To: Playback and Database Engines. This command copies items from the database engine to the Playback Engine.

Requests playback of the currently selected track or list of tracks. The currently selected tracks are placed in the Now Playing playlist, where they are optionally shuffled (if the shuffle feature is enabled). Finally, the specified track record index is passed to the player to play. Note that if the track index is  $-1$  (0xFFFFFFFF), the first track after the shuffle is complete is played first. If a track index of  $n$  is sent, the  $n$ th track of the selected tracks is played first, regardless of where it is located in the Now Playing playlist after the shuffle is performed (assuming that shuffle is on). In response, the iPod sends an ACK command indicating the status of the command.

**Note:** If the shuffle feature is enabled, any track that has the Skip When Shuffle attribute will be excluded from the Now Playing playlist unless its track record index is passed by this command, in which case it will play. With a playlist that contains only skip-when-shuffle tracks, the next action depends on the iPod model. With the iPhone and iPod touch, the next track will be shuffled and played; with other iPods, no further tracks will play.

**Table 4-58** PlayCurrentSelection command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x28	Command ID (bits 7:0)
6	0xNN	Selection track record index (bits 31:24)
7	0xNN	Selection track record index (bits 23:16)
8	0xNN	Selection track record index (bits 15:8)
9	0xNN	Selection track record index (bits 7:0)
10	0xNN	Packet payload checksum byte

## Command 0x0029: PlayControl

Direction: Device to iPod

Applies To: Playback Engine

This command is sent by the device to control the media playback state of the iPod. If the iPod is already in the requested state, the command has no effect and returns successfully. If the iPod does not enter the requested state successfully, an error status is returned. In response, the iPod sends an ACK command with the play control status.

**Note:** The iPod models before the 2G nano (09/2006) always return a successful status. Starting with the 2G nano, iPods return the actual play control status. This means that a next or previous track command will return an error if no media is playing.

**Table 4-59** PlayControl command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x29	Command ID (bits 7:0)
6	0xNN	Play control command code. See <a href="#">Table 4-60</a> (page 408).
7	0xNN	Packet payload checksum byte

[Table 4-60](#) (page 408) shows the iPod play states the device can set.

**Table 4-60** Play control command codes

Command	Code	Protocol	Comments
Reserved	0x00	Reserved	
Toggle Play/Pause	0x01	1.00	
Stop	0x02	1.00	
Next Track	0x03	1.00	These commands skip to the next or previous track. If the current track has played less than two seconds, Previous Track backs up to the beginning of the previous track. If the current track has played more than two seconds, it backs up to the beginning of the current track. These commands skip to the next or previous track even when the current track has chapters. To control audiobook playing by chapters, use the Next (0x08) and Previous (0x09) commands.
Previous Track	0x04	1.00	
Start FF	0x05	1.00	

Command	Code	Protocol	Comments
Start Rew	0x06	1.00	
End FF/Rew	0x07	1.00	
Next	0x08	1.06	If the current track is an audiobook or a podcast with chapters, these commands skip to the next or previous chapter; <b>see Note below.</b> Otherwise they act like Next Track (0x03) and Previous Track (0x04).
Previous	0x09	1.06	
Play	0x0A	1.13	
Pause	0x0B	1.13	
Next Chapter	0x0C	1.14	If the current track is an audiobook or a podcast with chapters, these commands skip to the next or previous chapter; otherwise they have no effect.
Previous Chapter	0x0D	1.14	
Reserved	0x0E – 0xFF	N/A	

**Note:** If a track has chapters, the Next code (0x08) will advance to the beginning of the next chapter. If the track is playing its last chapter or has no chapters, the Next code will advance to the beginning of the next track. If a track has chapters and is more than two seconds into the current chapter, the Previous code (0x09) will back up to the beginning of the current chapter. If it is less than two seconds into the current chapter, the Previous code will back up to the beginning of the previous chapter. If the track has no chapters, the Previous code will back up to the beginning of the previous track.

## Command: 0x002A: GetTrackArtworkTimes

Direction: Device to iPod

Applies To: Playback Engine

The device sends this command to the iPod to request the list of artwork time locations for a track. A 4-byte `trackIndex` specifies which track from the Playback Engine is to be selected. A 2-byte `formatID` indicates which type of artwork is desired. See "Transferring Album Art" (page 354).

The 2-byte `artworkIndex` specifies at which index to begin searching for artwork. A value of 0 indicates that the iPod should start with the first available artwork.

The 2-byte `artworkCount` specifies the maximum number of times (artwork locations) to be returned. A value of -1 (0xFFFF) indicates that there is no preferred limit. Note that podcasts may have a large number of associated images.

**Table 4-61** GetTrackArtworkTimes packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Comment
1	0x55	Start of packet
2	0x0D	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2A	Command ID (bits 7:0)
6	0xNN	trackIndex (31:24)
7	0xNN	trackIndex (23:16)
8	0xNN	trackIndex (15:8)
9	0xNN	trackIndex (7:0)
10	0xNN	formatID (15:8)
11	0xNN	formatID (7:0)
12	0xNN	artworkIndex (15:8)
13	0xNN	artworkIndex (7:0)
14	0xNN	artworkCount (15:8)
15	0xNN	artworkCount (7:0)
16	0xNN	Checksum

## Command: 0x002B: RetTrackArtworkTimes

---

Direction: iPod to Device

Applies To: Playback Engine

The iPod sends this command to the device to return the list of artwork times for a given track. The iPod returns zero or more 4-byte times, one for each piece of artwork associated with the track and format specified by `GetTrackArtworkTimes`. See ["Transferring Album Art"](#) (page 354).

The number of records returned will be no greater than the number specified in the `GetTrackArtworkTimes` command. It may, however, be less than requested. This can happen if there are fewer pieces of artwork available than were requested, or if the iPod is unable to place the full number in a single packet. Check the number of records returned against the results of `ReturnIndexedPlayingTrackInfo` with `infoType 8` to ensure that all artwork has been received.

**Table 4-62** RetTrackArtworkTimes packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Length of packet
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2B	Command ID (bits 7:0)
6	0xNN	time offset from track start in ms (31:24)
7	0xNN	time offset from track start in ms (23:16)
8	0xNN	time offset from track start in ms (15:8)
9	0xNN	time offset from track start in ms (7:0)
Preceding 4 bytes may be repeated NN times		
(last byte)	0xNN	Checksum

## Command 0x002C: GetShuffle

---

Direction: Device to iPod

Requests the current state of the iPod shuffle setting. The iPod responds with the ["Command 0x002D: ReturnShuffle"](#) (page 412) command.

**Table 4-63** GetShuffle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2C	Command ID (bits 7:0)
6	0xCD	Packet payload checksum byte

## Command 0x002D: ReturnShuffle

---

Direction: iPod to Device

Returns the current state of the shuffle setting. The iPod sends this command in response to the "[Command 0x002C: GetShuffle](#)" (page 411) command from the device.

**Table 4-64** ReturnShuffle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2D	Command ID (bits 7:0)
6	0xNN	Shuffle mode. See <a href="#">Table 4-65</a> (page 412).
7	0xNN	Packet payload checksum byte

[Table 4-65](#) (page 412) lists the possible values of the shuffle mode.

**Table 4-65** Shuffle modes

Value	Meaning
0x00	Shuffle off
0x01	Shuffle tracks
0x02	Shuffle albums
0x03 – 0xFF	Reserved

## Command 0x002E: SetShuffle

---

Direction: Device to iPod

Sets the iPod shuffle mode. The iPod shuffle modes are listed in [Table 4-65](#) (page 412). In response, the iPod sends an ACK command with the command status.



This command has an optional byte, byte 0x07, called the Restore on Exit byte. This byte can be used to restore the original shuffle setting in use when the accessory was attached to the iPod. A nonzero value restores the original shuffle setting of the iPod when the accessory is detached. If this byte is zero, the shuffle setting set by the accessory overwrites the original setting and persists after the accessory is detached from the iPod.

Accessory engineers should note that the shuffle mode affects items only in the Playback Engine. The shuffle setting does not affect the order of tracks in the database engine, so calling ["Command 0x001A: RetrieveCategorizedDatabaseRecords"](#) (page 393) on a database selection with the shuffle mode set returns a list of unshuffled tracks. To get the shuffled playlist, an accessory must query the Playback Engine by calling ["Command 0x0020: GetIndexedPlayingTrackTitle"](#) (page 398).

Shuffling tracks does not affect the track index, just the track at that index. If an unshuffled track at playback index 1 is shuffled, a new track is placed into index 1. The playback indexes themselves are not shuffled.

When shuffle mode is enabled, tracks that are marked “skip when shuffling” are filtered from the database selection. This affects all audiobooks and all tracks that the user has marked in iTunes. It also affects all podcasts unless their default “skip when shuffling” markings have been deliberately removed. To apply the filter to the Playback Engine, the accessory must send ["Command 0x0017: SelectDBRecord"](#) (page 390) or ["Command 0x0028: PlayCurrentSelection"](#) (page 407) after enabling shuffle mode.

**Note:** Accessory developers are encouraged to always use the Restore on Exit byte with a nonzero value to restore any settings modified by the accessory upon detach.

[Table 4-66](#) (page 413) shows a command to set the shuffle setting and optionally restore it on accessory detach.

**Table 4-66** SetShuffle command with Restore on Exit byte

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x05	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2E	Command ID (bits 7:0)
6	0xNN	New shuffle mode. See <a href="#">Table 4-65</a> (page 412).
7	0xNN	Restore on Exit byte. If 1, the original setting is restored on detach; if 0, the new setting persists after accessory detach.
8	0xNN	Packet payload checksum byte

[Table 4-67](#) (page 414) shows a command to make the shuffle setting persistent after the accessory detach.

**Table 4-67** SetShuffle command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2E	Command ID (bits 7:0)
6	0xNN	New shuffle mode. See <a href="#">Table 4-65</a> (page 412).
7	0xNN	Packet payload checksum byte

## Command 0x002F: GetRepeat

---

Direction: Device to iPod

Requests the track repeat state of the iPod. In response, the iPod sends a "[Command 0x0030: ReturnRepeat](#)" (page 414) command to the device.

**Table 4-68** GetRepeat command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x2F	Command ID (bits 7:0)
6	0xCA	Packet payload checksum byte

## Command 0x0030: ReturnRepeat

---

Direction: iPod to Device

Returns the current iPod track repeat state to the device. The iPod sends this command in response to the "[Command 0x002F: GetRepeat](#)" (page 414) command.

**Table 4-69** ReturnRepeat command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x30	Command ID (bits 7:0)
6	0xNN	Repeat state. See <a href="#">Table 4-70</a> (page 415).
7	0xNN	Packet payload checksum byte

[Table 4-70](#) (page 415) lists the possible values of the repeat state field.

**Table 4-70** Repeat state values

Value	Meaning
0x00	Repeat off
0x01	Repeat one track
0x02	Repeat all tracks
0x03 – 0xFF	Reserved

## Command 0x0031: SetRepeat

Direction: Device to iPod

Sets the repeat state of the iPod. The iPod track repeat modes are listed in [Table 4-70](#) (page 415). In response, the iPod sends an ACK command with the command status.

This command has an optional byte, byte 0x07, called the Restore on Exit byte. This byte can be used to restore the original repeat setting in use when the accessory was attached to the iPod. A nonzero value restores the original repeat setting of the iPod when the accessory is detached. If this byte is zero, the repeat setting set by the accessory overwrites the original setting and persists after the accessory is detached from the iPod.

**Note:** Accessory developers are encouraged to always use the Restore on Exit byte with a nonzero value to restore any settings modified by the accessory upon detach.

[Table 4-71](#) (page 416) shows a command to set the repeat setting and optionally restore it on accessory detach.

**Table 4-71** SetRepeat command with Restore on Exit byte

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x05	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x31	Command ID (bits 7:0)
6	0xNN	New repeat state. See <a href="#">Table 4-70</a> (page 415).
7	0xNN	Restore on Exit byte. If 1, the original setting is restored on detach; if 0, the new setting persists after accessory detach.
8	0xNN	Packet payload checksum byte

[Table 4-72](#) (page 416) shows a command to make the repeat setting persistent after the accessory detach.

**Table 4-72** SetRepeat command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x31	Command ID (bits 7:0)
6	0xNN	New repeat state. See <a href="#">Table 4-70</a> (page 415).
7	0xNN	Packet payload checksum byte

## Command 0x0032: SetDisplayImage

Direction: Device to iPod

Sets a bitmap image that is shown on the iPod display when it is connected to the device. The intent is to allow third party branding when the iPod is communicating with an external device. An image downloaded using this mechanism replaces the default checkmark bitmap image that is displayed when iPod is connected

to an external device. The new bitmap is retained in RAM for as long as the iPod remains powered. After a system reset or Sleep state, the new bitmap is lost and the checkmark image restored as the default when the iPod next enters Extended Interface mode after power-up.

Before setting a monochrome display image, the device can send the "[Command 0x0033: GetMonoDisplayImageLimits](#)" (page 422) command to obtain the current iPod display width, height and pixel format. The monochrome display information returned in the "[Command 0x0034: ReturnMonoDisplayImageLimits](#)" (page 422) command can be useful to the device in deciding which type of display image format is suitable for downloading to the iPod.

On iPods with color displays, devices can send the "[Command 0x0039: GetColorDisplayImageLimits](#)" (page 427) command to obtain the iPod color display width, height, and pixel formats. The color display information is returned in the "[Command 0x003A: ReturnColorDisplayImageLimits](#)" (page 428) command.

To set a display image, the device must successfully send `SetDisplayImage` descriptor and data commands to the iPod. The `SetDisplayImage` descriptor command (packet index 0x0000) must be sent first, as it gives the iPod a description of the image to be downloaded. This command is shown in [Table 4-73](#) (page 417)). The image descriptor command includes image pixel format, image width and height, and display row size (stride) in bytes. Optionally, the descriptor command may also contain the beginning data of the display image, as long as it remains within the maximum length limits of the packet.

Following the descriptor command, the `SetDisplayImage` data commands (packet index 0x0001 – 0xNNNN) must be sent using sequential packet indices until the entire image has been sent to the iPod.

**Note:** The `SetDisplayImage` command payload length is limited to 500 bytes. This packet length limit and the size and format of the display image generally determine the minimum number of packets that are required to set a display image.

**Note:** Starting with the second generation iPod nano with version 1.1.2 firmware, using the `SetDisplayImage` command is limited to once every 15 seconds over USB transport. The iPod classic and iPod 3G nano apply a different restriction to both USB and UART transports: calls made to `SetDisplayImage` after the first 15 seconds will return a successful ACK command, but the bitmap will not be displayed on the iPod's screen. Hence, use of the `SetDisplayImage` command should be limited to drawing bitmap images only immediately after entering Extended mode. The iPod touch accepts the `SetDisplayImage` command but will not draw it on the iPod's screen.

[Table 4-73](#) (page 417) shows the format of a descriptor command. This example assumes the display image descriptor data exceeds the small packet payload capacity; a large packet format is shown.

**Table 4-73** `SetDisplayImage` descriptor command (packet index = 0x0000)

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x00	Packet payload marker (large format)
3	0xNN	Large packet payload length (bits 15:8)
4	0xNN	Large packet payload length (bits 7:0)

Byte number	Value	Meaning
5	0x04	Lingo ID: Extended Interface lingo
6	0x00	Command ID (bits 15:8)
7	0x32	Command ID (bits 7:0)
8	0x00	Descriptor command index (bits 15:8). These fields uniquely identify each packet in the <code>SetDisplayImage</code> transaction. The first command is the Descriptor command and always starts with an index of 0x0000.
9	0x00	Descriptor command index (bits 7:0)
10	0xNN	Display pixel format code. See <a href="#">Table 4-75</a> (page 420).
11	0xNN	Image width in pixels (bits 15:8). The number of pixels, from left to right, per row.
12	0xNN	Image width in pixels (bits 7:0)
13	0xNN	Image height in pixels (bits 15:8). The number of rows, from top to bottom, in the image.
14	0xNN	Image height in pixels (bits 7:0)
15	0xNN	Row size (stride) in bytes (bits 31:24). The number of bytes representing one row of pixels. Each row is zero-padded to end on a 32-bit boundary. The cumulative size, in bytes, of the image data, transferred across all packets in this transaction is effectively $(\text{Row Size} * \text{Image Height})$ .
16	0xNN	Row size (stride) in bytes (bits 23:16)
17	0xNN	Row size (stride) in bytes (bits 15:8)
18	0xNN	Row size (stride) in bytes (bits 7:0)
19 – N	0xNN	Display image pixel data
(last byte)	0xNN	Packet payload checksum byte

**Note:** [Table 4-74](#) (page 418) shows the format of a data command. This example assumes the display image data exceeds the small packet payload capacity; a large packet format is shown.

**Table 4-74** `SetDisplayImage` data packet (packet index = 0x0001 – 0xNNNN)

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x00	Packet payload marker (large format)

Byte number	Value	Meaning
3	0xNN	Large packet payload length (bits 15:8)
4	0xNN	Large packet payload length (bits 7:0)
5	0x04	Lingo ID: Extended Interface lingo
6	0x00	Command ID (bits 15:8)
7	0x32	Command ID (bits 7:0)
8	0xNN	Descriptor command index (bits 15:8). These fields uniquely identify each packet in the <code>SetDisplayImage</code> transaction. The first command is the descriptor command, shown in <a href="#">Table 4-73</a> (page 417). The remaining $n - 1$ commands are simply data packets, where $n$ is determined by the size of the image.
9	0xNN	Descriptor packet index (bits 7:0)
10 – $N$	0xNN	Display image pixel data
(last byte)	0xNN	Packet payload checksum byte

**Note:** A known issue causes `SetDisplayImage` data packet lengths less than 11 bytes to return a bad parameter error (0x04) ACK on 3G iPods.

The iPod display is oriented as a rectangular grid of pixels. In the horizontal direction (x-coordinate), the pixel columns are numbered, left to right, from 0 to  $C_{max}$ . In the vertical direction (y-coordinate), the pixel rows are numbered, top to bottom, from 0 to  $R_{max}$ . Therefore, an (x,y) coordinate of (0,0) represents the upper-leftmost pixel on the display and ( $C_{max}, R_{max}$ ) represents the lower-rightmost pixel on the display. A portion of the iPod display pixel layout is shown below, where x is the column number, y is the row number, and ( $C_{max}, R_{max}$ ) represents the maximum row and column numbers.

Pixel layout	x									
y	0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0	...	$C_{max},0$
	0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1	...	$C_{max},1$
	0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2	...	$C_{max},2$
	0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3	...	$C_{max},3$
	0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4	...	$C_{max},4$
	0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5	...	$C_{max},5$
	0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6	...	$C_{max},6$
	0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7	...	$C_{max},7$
	...	...	...	...	...	...	...	...	...	...

	0, Rmax	1, Rmax	2, Rmax	3, Rmax	4, Rmax	5, Rmax	6, Rmax	7, Rmax	...	Cmax,Rmax
--	---------	---------	---------	---------	---------	---------	---------	---------	-----	-----------

The iPod display pixel formats are shown in [Table 4-75](#) (page 420).

**Table 4-75** Display pixel format codes

Display pixel format	Code	Protocol version
Reserved	0x00	N/A
Monochrome, 2 bits per pixel	0x01	1.01
RGB 565 color, little-endian, 16 bpp	0x02	1.09
RGB 565 color, big-endian, 16 bpp	0x03	1.09
Reserved	0x04 – 0xFF	N/A

iPods with color screens support all three image formats. All other iPods support only display pixel format 0x01 (monochrome, 2 bpp).

### Display Pixel Format 0x01

Display pixel format 0x01 (monochrome, 2 bits per pixel) is the pixel format supported by all iPods. Each pixel consists of 2 bits that control the pixel intensity. The pixel intensities and associated binary codes are listed in [Table 4-76](#) (page 420).

**Table 4-76** 2 bpp monochrome pixel intensities

Pixel Intensity	Binary Code
Pixel off (not visible)	00b
Pixel on 25% (light grey)	01b
Pixel on 50% (dark grey)	10b
Pixel on 100% (black)	11b

Each byte of image data contains four packed pixels. The pixel ordering within bytes and the byte ordering within 32 bits is shown below.

Image Data Byte 0x0000							
Pixel 0		Pixel 1		Pixel 2		Pixel 3	
7	6	5	4	3	2	1	0



Image Data Byte 0x0001							
Pixel 4		Pixel 5		Pixel 6		Pixel 7	
7	6	5	4	3	2	1	0

Image Data Byte 0x0002							
Pixel 8		Pixel 9		Pixel 10		Pixel 11	
7	6	5	4	3	2	1	0

Image Data Byte 0x0003							
Pixel 12		Pixel 13		Pixel 14		Pixel 15	
7	6	5	4	3	2	1	0

Image Data Byte 0xNNNN							
Pixel N		Pixel N+1		Pixel N+2		Pixel N+3	
7	6	5	4	3	2	1	0

### Display Pixel Formats 0x02 and 0x03

Display pixel format 0x02 (RGB 565, little-endian) and display pixel format 0x03 (RGB 565, big-endian) are available for use in all iPods with color screens. Each pixel consists of 16 bits that control the pixel intensity for the colors red, green, and blue.

It takes two bytes to represent a single pixel. Red is represented by 5 bits, green is represented by 6 bits, and blue by the final 5 bits. A 32-bit sequence represents 2 pixels. The pixel ordering within bytes and the byte ordering within 32 bits for display format 0x02 (RGB 565, little-endian) is shown below.

Image Data Byte 0x0000							
Pixel 0, lower 3 bits of green				Pixel 0, all 5 bits of blue			
7	6	5	4	3	2	1	0

Image Data Byte 0x0001							
Pixel 0, all 5 bits of red				Pixel 0, upper 3 bits of green			
7	6	5	4	3	2	1	0

Image Data Byte 0x0002							
Pixel 1, lower 3 bits of green				Pixel 1, all 5 bits of blue			
7	6	5	4	3	2	1	0

Image Data Byte 0x0003							
Pixel 1, all 5 bits of red				Pixel 1, upper 3 bits of green			
7	6	5	4	3	2	1	0

The format for display pixel format 0x03 (RGB 565, big-endian, 16 bpp) is almost identical, with the exception that bytes 0 and 1 are swapped and bytes 2 and 3 are swapped.

## Command 0x0033: GetMonoDisplayImageLimits

Direction: Device to iPod

Requests the limiting characteristics of the monochrome image that can be sent to the iPod for display while it is connected to the device. It can be used to determine the display pixel format and maximum width and height of a monochrome image to be set using the "[Command 0x0032: SetDisplayImage](#)" (page 416) command. In response, the iPod sends a "[Command 0x0034: ReturnMonoDisplayImageLimits](#)" (page 422) command to the device with the requested display information. The `GetMonoDisplayImageLimits` command is supported by iPods with either monochrome or color displays. To obtain color display image limits, use "[Command 0x0039: GetColorDisplayImageLimits](#)" (page 427).

**Table 4-77** `GetMonoDisplayImageLimits` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x33	Command ID (bits 7:0)
6	0xC6	Packet payload checksum byte

## Command 0x0034: ReturnMonoDisplayImageLimits

Direction: iPod to Device

Returns the limiting characteristics of the monochrome image that can be sent to the iPod for display while it is connected to the device. The iPod sends this command in response to the "[Command 0x0033: GetMonoDisplayImageLimits](#)" (page 422) command. Monochrome display characteristics include maximum image width and height and the display pixel format.

**Table 4-78** ReturnMonoDisplayImageLimits command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x34	Command ID (bits 7:0)
6	0xNN	Maximum image width in pixels (bits 15:8)
7	0xNN	Maximum image width in pixels (bits 7:0)
8	0xNN	Maximum image height in pixels (bits 15:8)
9	0xNN	Maximum image height in pixels (bits 7:0)
10	0xNN	Display pixel format (see <a href="#">Table 4-75</a> (page 420)).
11	0xNN	Packet payload checksum byte

## Command 0x0035: GetNumPlayingTracks

Direction: Device to iPod

Applies To: Playback Engine

Requests the number of tracks in the list of tracks queued to play on the iPod. In response, the iPod sends a "[Command 0x0036: ReturnNumPlayingTracks](#)" (page 424) command with the count of tracks queued to play.

**Table 4-79** GetNumPlayingTracks command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo

Byte number	Value	Meaning
4	0x00	Command ID (bits 15:8)
5	0x35	Command ID (bits 7:0)
6	0xC4	Packet payload checksum byte

## Command 0x0036: ReturnNumPlayingTracks

---

Direction: iPod to Device

Returns the number of tracks in the actual list of tracks queued to play, including the currently playing track (if any). The iPod sends this command in response to the ["Command 0x0035: GetNumPlayingTracks"](#) (page 423) command.

**Table 4-80** ReturnNumPlayingTracks command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x36	Command ID (bits 7:0)
6	0xNN	Number of tracks playing (bits 31:24)
7	0xNN	Number of tracks playing (bits 23:16)
8	0xNN	Number of tracks playing (bits 15:8)
9	0xNN	Number of tracks playing (bits 7:0)
10	0xNN	Packet payload checksum byte

## Command 0x0037: SetCurrentPlayingTrack

---

Direction: Device to iPod

Applies To: Playback Engine

Sets the index of the track to play in the Now Playing playlist on the iPod. The index that is specified here is obtained by sending the "[Command 0x0035: GetNumPlayingTracks](#)" (page 423) and "[Command 0x001E: GetCurrentPlayingTrackIndex](#)" (page 397) commands to obtain the number of playing tracks and the current playing track index, respectively. In response, the iPod sends an ACK command indicating the status of the command.

**Note:** This command is usable only when the iPod is in a playing or paused state. If the iPod is stopped, this command fails. If this command is sent with the current playing track index, the iPod pauses playback momentarily and then resumes. If the current playing track has index 1 and this command passes index 1, playback does not restart from the beginning.

**Table 4-81** SetCurrentPlayingTrack command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x07	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x37	Command ID (bits 7:0)
6	0xNN	New current playing track index (bits 31:24)
7	0xNN	New current playing track index (bits 23:16)
8	0xNN	New current playing track index (bits 15:8)
9	0xNN	New current playing track index (bits 7:0)
10	0xNN	Packet payload checksum byte

## Command 0x0038: SelectSortDBRecord

Direction: Device to iPod

Applies To: Database Engine

Selects one or more records in the iPod database, based on a category-relative index. For example, selecting category 2 (Artist), record index 1, and sort order 3 (Album) results in a list of selected tracks (records) from the second artist in the artist list, sorted by album name. Selections are additive and limited by the category hierarchy; see "[Database Category Hierarchies](#)" (page 350). Subsequent selections are made based on the subset of records resulting from previous selections and not from the entire database. The database category types are shown in [Table 4-37](#) (page 391). The sort order options and codes are shown in [Table 4-83](#) (page 426).

**Table 4-82** SelectSortDBRecord command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x09	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x38	Command ID (bits 7:0)
6	0xNN	Database category type. See <a href="#">Table 4-37</a> (page 391).
7	0xNN	Category record index (bits 31:24)
8	0xNN	Category record index (bits 23:16)
9	0xNN	Category record index (bits 15:8)
10	0xNN	Category record index (bits 7:0)
11	0xNN	Database sort type. See <a href="#">Table 4-83</a> (page 426).
12	0xNN	Packet payload checksum byte

[Table 4-83](#) (page 426) shows the possible sort orders.

**Table 4-83** Database sort order options

Sort Order	Code	Protocol version
Sort by genre	0x00	1.00
Sort by artist	0x01	1.00
Sort by composer	0x02	1.00
Sort by album	0x03	1.00
Sort by name	0x04	1.00
Reserved	0x05	N/A
Sort by release date	0x06	1.08
Sort by series (video only)	0x07	1.12
Sort by season (video only)	0x08	1.12
Sort by episode (video only)	0x09	1.12

Sort Order	Code	Protocol version
Reserved	0x0A – 0xFE	N/A
Use default sort type	0xFF	1.00

The default order of song and audiobook tracks on the iPod is alphabetical by artist, then alphabetical by that artist's albums, then ordered according to the order of the tracks on the album. For podcasts, the default order of episode tracks is reverse chronological—that is, the newest ones are first—then alphabetical by podcast name.

The `SelectSortDBRecord` command can be used to sort all the song and audiobook tracks on the iPod alphabetically as follows:

1. ["Command 0x0016: ResetDBSelection"](#) (page 389)
2. ["Command 0x0018: GetNumberCategorizedDBRecords"](#) (page 392) for the Playlist category.
3. `SelectSortDBRecord` based on the Playlist category, using a record index of 0 to select the All Tracks playlist and the sort by name (0x04) sort order.
4. `GetNumberCategorizedDBRecords` for the Track category.
5. ["Command 0x001A: RetrieveCategorizedDatabaseRecords"](#) (page 393) based on the Track category, using a start index of 0 and an end index of the number of records returned by the call to `GetNumberCategorizedDBRecords` in step 4.

The sort order of artist names ignores certain articles such that the artist "The Doors" is sorted under the letter 'D' and not 'T'; this matches the behavior of iTunes. The sort order is different depending on the language setting used in the iPod. The list of ignored articles may change in the future without notice.

The `SelectDBRecord` command may also be used to select a database record with the default sort order.

**Note:** The sort order field is ignored for the Audiobook category. Audiobooks are automatically sorted by track title. The sort order for podcast tracks defaults to release date, with the newest track coming first.

## Command 0x0039: GetColorDisplayImageLimits

Direction: Device to iPod

Requests the limiting characteristics of the color image that can be sent to the iPod for display while it is connected to the device. It can be used to determine the display pixel format and maximum width and height of a color image to be set using the ["Command 0x0032: SetDisplayImage"](#) (page 416) command. In response, the iPod sends a ["Command 0x003A: ReturnColorDisplayImageLimits"](#) (page 428) command to the device with the requested display information. This command is supported only by iPods with color displays.

**Table 4-84** GetColorDisplayImageLimits command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x39	Command ID (bits 7:0)
6	0xC0	Packet payload checksum byte

## Command 0x003A: ReturnColorDisplayImageLimits

Direction: iPod to Device

Returns the limiting characteristics of the color image that can be sent to the iPod for display while it is connected to the device. The iPod sends this command in response to the ["Command 0x0039: GetColorDisplayImageLimits"](#) (page 427) command. Display characteristics include maximum image width and height and the display pixel format.

**Note:** If the iPod supports multiple display image formats, a five byte block of additional image width, height, and pixel format information is appended to the payload for each supported display format. The list of supported color display image formats returned by the iPod may change in future software versions. Devices must be able to parse a variable length list of supported color display formats to search for compatible formats.

**Table 4-85** ReturnColorDisplayImageLimits command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3A	Command ID (bits 7:0)
6	0xNN	Maximum image width in pixels (bits 15:8)
7	0xNN	Maximum image width in pixels (bits 7:0)



Byte number	Value	Meaning
8	0xNN	Maximum image height in pixels (bits 15:8)
9	0xNN	Maximum image height in pixels (bits 7:0)
10	0xNN	Display pixel format (see <a href="#">Table 4-75</a> (page 420)).
11 - N	0xNN	Optional display image width, height, and pixel format for the second to nth supported display formats, if present.
(last byte)	0xNN	Packet payload checksum byte

## Command 0x003B: ResetDBSelectionHierarchy

Direction: Device to iPod

Applies To: Database Engine

This command carries a single byte in its payload (byte 6). A hierarchy selection value of 0x01 means that the accessory wants to navigate the audio hierarchy; a hierarchy selection value of 0x02 means that the accessory wants to navigate the video hierarchy.

**Note:** If the device sends a `ResetDBSelectionHierarchy` command while selecting the audio hierarchy, the database resets itself to the audio hierarchy and any video database selections are invalidated. Video selections already passed to the Playback Engine are unaffected.

**Table 4-86** `ResetDBSelectionHierarchy` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3B	Command ID (bits 7:0)
6	0x01 or 0x02	Hierarchy selection
7	0xNN	Packet payload checksum byte

**Note:** The iPod will return an error if a device attempts to enable an unsupported hierarchy, such as a video hierarchy on an iPod model that does not support video.

## Command 0x003C: GetDBiTunesInfo

Direction: Dev to iPod

Applies to Database Engine.

This command is sent by the device to get the specified iPod iTunes database metadata information. In response, the iPod sends a `RetDBiTunesInfo` command with the requested metadata.

**Note:** This DB metadata information is updated when the iPod enters remote UI mode. This ensures that the information is updated following an iTunes sync event.

**Table 4-87** `GetDBiTunesInfo` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x04	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3C	Command ID (bits 7:0)
6	0xNN	iTunes database metadata type (see <a href="#">Table 4-88</a> (page 430))
7	0xNN	Packet payload checksum byte

[Table 4-88](#) (page 430) shows the iTunes database metadata type codes.

**Table 4-88** iTunes database metadata types

Type code	Description
0x00	Database UID unique to an iPod and assigned by iTunes. This ID does not change when the iPod is synced with iTunes nor when the iPod's content is changed. It is not the same as the track UID returned by <code>GetDBTrackInfo</code> or <code>GetPBTrackInfo</code> and used by <code>GetUIDTrackInfo</code> .
0x01	Last sync date/time to iTunes on computer; updated when the iPod is synced with iTunes even if no content has been added or deleted.
0x02	Total audio track count (including audio podcasts and audiobooks)

Type code	Description
0x03	Total video track count (including movies, TV series, and video podcasts)
0x04	Total audiobook count
0x05	Total photo count
0x06-0xFF	Reserved

## Command 0x003D: RetDBiTunesInfo

Direction: iPod to Dev

Applies to Database Engine.

This command is returned by the iPod in response to a `GetDBiTunesInfo` command received from the device. If the requested iTunes metadata information type is not within the valid range, an `ACK` command with a bad parameter status is returned.

**Table 4-89** RetDBiTunesInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3D	Command ID (bits 7:0)
6	0xNN	iTunes database metadata type (see <a href="#">Table 4-88</a> (page 430))
7...	...	iTunes database metadata information (see <a href="#">Table 4-90</a> (page 431))
(last byte)	0xNN	Packet payload checksum byte

[Table 4-90](#) (page 431) shows the iTunes database metadata information formats.

**Table 4-90** iTunes database metadata information formats

Type code	Bytes	Description
0x00	8	iTunes database unique 64-bit identifier
0x01	7	Last sync date/time (see <a href="#">Table 4-91</a> (page 432))

Type code	Bytes	Description
0x02	4	Total audio track count
0x03	4	Total video track count
0x04	4	Total audiobook count
0x05	4	Total photo count
0x06-0xFF	Reserved	

**Table 4-91** Date/time format

Byte	Description	Values
0	Seconds	00 - 59
1	Minute	00 - 59
2	Hour	00 = 12am, 23 = 11pm
3	Day	01 = 1st, 31 = 31st
4	Month	01 = Jan, 12 = Dec
5-6	Year	2007 = year 2007

## Command 0x003E: GetUIDTrackInfo

Direction: Dev to iPod

Applies to Database Engine.

This command is sent by the device to get one or more types of track information using the track's iPod-unique identifier. In response, the iPod returns separate `RetUIDTrackInfo` commands for each type of track information requested. The track information mask is variable length; it is not necessary to transmit any trailing bytes in which no bits are set. If the track information mask contains any unrecognized track information type bits, a single `ACK` command is returned with a bad parameter status.

**Note:** The device's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the device in rapid succession, without any pauses or interruptions.

**Table 4-92** `GetUIDTrackInfo` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet

Byte number	Value	Meaning
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3E	Command ID (bits 7:0)
6–13	0xNNNNNNNNNNNNNNNN	Unique track identifier; a device can obtain this value by sending a <a href="#">GetDBTrackInfo</a> (page 437) or <a href="#">GetPBTrackInfo</a> (page 439) command with bit 7 of byte 14 set to 1.
14...	...	Track information type bitmask (see <a href="#">Table 4-93</a> (page 433)); do not set bit 7.
(last byte)	0xNN	Packet payload checksum byte

[Table 4-93](#) (page 433) lists the track information type bitmask bits.

**Table 4-93** Track information type bits

Bit	Description
0	Capabilities (media kind, skip when shuffle, has artwork, has bookmark, has lyrics, is audiobook, etc.)
1	Track name
2	Artist name
3	Album name
4	Genre name
5	Composer name
6	Total track time duration
7	Unique track identifier
8	Chapter count
9	Chapter times
10	Chapter names
11	Lyrics of the song currently playing in the Playback Engine
12	Description
13	Album track index
14	Disc set album index

Bit	Description
15	Play count
16	Skip count
17	Podcast release date
18	Last played date/time
19	Year (release date)
20	Star rating
21	Series name
22	Season number
23	Track volume adjust
24	Track EQ preset
25	Track sample rate
26	Bookmark offset
27	Start/stop time offset
28...	Reserved

**Note:** Requesting the lyrics of a track not currently being played will result in a null string being returned. Lyrics for a track being played may take up to 5 seconds to return. Use `SetPlayStatusChangeNotification` to be notified when the lyrics for the currently playing track become available.

## Command 0x003F: RetUIDTrackInfo

Direction: iPod to Dev

Applies to Database Engine.

This command is sent by the iPod in response to a `GetUIDTrackInfo` command received from the device. Multiple responses may be sent for each type of track information requested (e.g., chapter names).

**Note:** The device's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the device in rapid succession, without any pauses or interruptions.

**Table 4-94** RetUIDTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x3F	Command ID (bits 7:0)
6–13	0xNNNNNNNNNNNNNNNN	Unique track identifier
14	0xNN	Track information type (bit position number)
15...	...	Track information data (see <a href="#">Table 4-95</a> (page 435))
(last byte)	0xNN	Packet payload checksum byte

[Table 4-95](#) (page 435) shows the track information data formats.

**Table 4-95** Track information data formats

Type	Description	Bytes	Format
0	Capabilities	4	See <a href="#">Table 4-96</a> (page 437)
1	Track name	NN	Null-terminated UTF8 string
2	Artist name	NN	Null-terminated UTF8 string
3	Album name	NN	Null-terminated UTF8 string
4	Genre name	NN	Null-terminated UTF8 string
5	Composer name	NN	Null-terminated UTF8 string
6	Total track duration	4	Milliseconds
7	iTunes unique track ID	8	An 8-byte UID that uniquely identifies an iPod track. This track UID is different from the iTunes database UID returned by RetDBiTunesInfo.
8	Chapter count	2	Chapter count (0 = no chapters)

Type	Description	Bytes	Format
9	Chapter times	2,4	2 bytes chapter index (0 = first chapter) followed by 4 bytes chapter offset in milliseconds from beginning of track. A separate index/offset pair is appended for each track chapter. If a track has no chapters, no data is returned.
10	Chapter names	2,NN	2 bytes chapter index (0 = first chapter) followed by the chapter name as null-terminated UTF8 string. A separate index/name pair is appended for each track chapter. If a track has no chapters, no data is returned.
11	Lyrics of the song currently playing in the Playback Engine	2,2,NN	2 bytes current track lyrics section index (0 = first section, 0xNNNN = last section index), followed by 2 bytes maximum track lyrics section index (0 = only 1 section, 0xNNNN = maximum section index), followed by some or all of the track lyrics string. The track lyrics as a whole consists of a single null-terminated UTF8 string. If the lyrics string is too long to be carried in a single packet, then the string is broken into sections and carried in separate packets, with different values for each section index. The last lyrics packet section contains the null terminator character for the full string. Lyrics sections before the last section do not include null terminators and may not be valid UTF8 strings. Accessories must use the packet payload length to determine the length of the track lyrics string and assemble it by concatenating its substrings in order. Requesting the lyrics of a track not currently being played will result in a null string being returned.
12	Description	NN	Null-terminated UTF8 string
13	Album track index	2	index number
14	Disc set album index	2	index number
15	Play count	4	Track play count (0 = track not played)
16	Skip count	4	Track skip count (0 = track not skipped)
17	Podcast release date	7	Date/time (see <a href="#">Table 4-91</a> (page 432))
18	Last played date/time	7	Date/time (see <a href="#">Table 4-91</a> (page 432)); all zeroes if the track has never been played.
19	Year (release date)	2	Year in which track was released
20	Star rating	1	Star rating of track; 00 = No stars, 20 = 1 star, 40 = 2 stars, 60 = 3 stars, 80 = 4 stars, 100 = 5 stars.
21	Series name	NN	Null-terminated UTF8 string
22	Season number	2	Season number (1 = First season)
23	Track volume adjust	1	Track volume attenuation/amplification adjustment (0x9C = -100%, 0x00 = no adjust, 0x64 = +100%)



Type	Description	Bytes	Format
24	Track EQ preset	2	Track equalizer preset index
25	Data rate	4	Track bit rate (kilobits per second)
26	Bookmark offset	4	Bookmark offset from start of track in milliseconds
27	Start/stop time offset	4,4	4 bytes start time followed by 4 bytes stop time in milliseconds
28...	Reserved		

**Table 4-96** Capabilities bits

Bit	Description
00	1 = Is audiobook
01	1 = Has chapters
02	1 = Has artwork
03	1 = Has lyrics
04	1 = Is podcast episode
05	1 = Has release date
06	1 = Has description
07	1 = Is video
08	1 = Is queued as video (in the Playback Engine only, not in the database)
31:09	Reserved

## Command 0x0040: GetDBTrackInfo

---

Direction: Dev to iPod

Applies to Database Engine.

This command is sent by the device to get the specified iPod database track information types for the specified track index range. In response, the iPod returns separate `RetDBTrackInfo` packets for each type of track information requested. The starting track index is based on the current database track selection(s). A track count of 0xFFFFFFFF (–1) returns the track information for all iPod tracks from the starting DB track index. The track information mask is variable length; it is not necessary to transmit any trailing bytes in which no track information bits are set. If the track index, count, or information mask contains any invalid data, a single ACK command is returned with a bad parameter status.

**Note:** The device's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the device in rapid succession, without any pauses or interruptions.

**Table 4-97** GetDBTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x40	Command ID (bits 7:0)
6–9	0xNNNNNNNN	Track database start index
10–13	0xNNNNNNNN	Track count (from track start index)
14...	...	Track information type bitmask (see <a href="#">Table 4-93</a> (page 433))
(last byte)	0xNN	Packet payload checksum byte

**Note:** Requesting the lyrics of a track not currently being played will result in a null string being returned. Lyrics for a track being played may take up to 5 seconds to return. Use `SetPlayStatusChangeNotification` to be notified when the lyrics for the currently playing track become available.

## Command 0x0041: RetDBTrackInfo

Direction: iPod to Dev

Applies to Database Engine.

This command is sent by the iPod in response to a `GetDBTrackInfo` command received from the device. Multiple responses may be sent for each type of track information requested (e.g., chapter names).

**Note:** The device's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the device in rapid succession, without any pauses or interruptions.

**Table 4-98** RetDBTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x41	Command ID (bits 7:0)
6–9	0xNNNNNNNN	Track database index
10	0xNN	Track information type (bit position number)
11...	...	Track information data (see <a href="#">Table 4-95</a> (page 435))
(last byte)	0xNN	Packet payload checksum byte

## Command 0x0042: GetPBTrackInfo

Direction: Dev to iPod

Applies to Playback Engine.

This command is sent by the device to get the specified iPod playing track information types for the specified track index range. In response, the iPod returns separate RetPBTrackInfo packets for each type of track information requested. A track count of 0xFFFFFFFF (–1) returns the track information for all iPod tracks from the starting PB track index. The track information mask is variable length; it is not necessary to transmit any trailing bytes in which no track information bits are set. If the track index, count, or information mask contains any invalid data, a single ACK command is returned with a bad parameter status.

**Note:** The device's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the device in rapid succession, without any pauses or interruptions.

**Table 4-99** GetPBTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)

Byte number	Value	Meaning
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x42	Command ID (bits 7:0)
6–9	0xNNNNNNNN	Track playing start index
10–13	0xNNNNNNNN	Track count (from track start index)
14...	...	Track information type bitmask (see <a href="#">Table 4-93</a> (page 433))
(last byte)	0xNN	Packet payload checksum byte

**Note:** Requesting the lyrics of a track not currently being played will result in a null string being returned. Lyrics for a track being played may take up to 5 seconds to return. Use `SetPlayStatusChangeNotification` to be notified when the lyrics for the currently playing track become available.

## Command 0x0043: RetPBTrackInfo

Direction: iPod to Dev

Applies to Playback Engine.

This command is sent by the iPod in response to a `GetPBTrackInfo` command received from the device. Multiple responses may be sent for each type of track information requested (e.g., chapter names).

**Note:** The device's input buffer must be of sufficient size to accept all requested track information response packets without errors. The requested packets may be sent to the device in rapid succession, without any pauses or interruptions.

**Table 4-100** RetPBTrackInfo command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)

Byte number	Value	Meaning
5	0x43	Command ID (bits 7:0)
6–9	0xNNNNNNNN	Track playback index
10	0xNN	Track information type (bit position number)
11...	...	Track information data (see <a href="#">Table 4-95</a> (page 435))
(last byte)	0xNN	Packet payload checksum byte

## Known Issues

- iPods may take up to 20 seconds to detect that the remote device has been detached from the 30-pin connector. This depends on the firmware version and whether or not power is being supplied to the iPod.
- External devices such as simple remote controls may still be active, even when the iPod has entered Extended Interface mode operation. Users should be discouraged from attaching any devices to the iPod while it is in Extended Interface mode.
- iPod mini version 1.4 and 4G iPod version 3.1 do not return audiobook chapter names. They return an empty string instead.

## Protocol History

The following is a history of the changes made to the iPod Extended Interface protocol.

**Table 4-101** History of the iPod Extended Interface protocol

Version	Changes
<b>Version 1.14</b>	Added status change control extensions to <code>SetPlayStatusChangeNotification</code> and <code>PlayStatusChangeNotification</code> .
	Added new support for Next Chapter and Previous Chapter in the <code>PlayControl</code> command.
<b>Version 1.13</b>	Added support for nested playlists.
	Added Play and Pause control codes to command 0x0029.
	Added new commands 0x003C through 0x0043.
<b>Version 1.12</b>	Improved video browsing support for <code>SelectDBRecord(-1)</code> .
<b>Version 1.11</b>	This version adds <code>ResetDBSelectionHierarchy</code> command to enable video browsing.

Version	Changes
	New feature: video browsing.
	Chapter information can be retrieved for all tracks in the Now Playing list, not just for the currently playing track.
<b>Version 1.10</b>	This version adds commands 0x000E through 0x0011 and 0x002A through 0x002B.
	New features: Code restructured to improve performance.
	Added indexed playing track genre and composer info.
	Added playing track artwork support.
	Bug fixes: Return track description and lyrics info if present.
	Notify track changes even when playback is paused.
<b>Version 1.09</b>	This version adds support for color images, podcast chapters, and fixes a few bugs:
	<code>SetDisplayImage</code> supports color images on color iPods.
	A bug that caused <code>SelectDBRecord</code> to fail if there were no podcasts on the iPod is fixed.
	A bug that caused <code>SelectDBRecord</code> and <code>SelectSortDBRecord</code> to reverse track order after a podcast was selected is fixed.
	<code>PlayStatusChange</code> notifications support podcast chapters.
	<code>SetCurrentPlayingChapter</code> now accepts a chapter index for podcasts.
	Added new commands <code>GetColorDisplayImageLimits</code> and <code>ReturnColorDisplayImageLimits</code> to obtain information about color display images.
<b>Version 1.08</b>	This version adds support for the Podcast category, chapter names, and indexed playing track information. It also includes a bug fix for <code>SetDisplayImage</code> .
<b>Version 1.07</b>	This version adds the restore on exit feature to the set shuffle, set repeat, and set audiobook speed setting commands. It also fixes a few bugs in the database engine management commands.
	<code>ResetDBSelection</code> clears the database sort order.
	<code>SelectDBRecord</code> with an invalid index returns a command error.
	Audiobook speed restore on exit with optional flag.
	Shuffle setting restore on exit with optional flag.
	Repeat setting restore on exit with optional flag.

Version	Changes
<b>Version 1.06</b>	This version adds several new lingo 0x04 commands to allow remote accessories to control audiobook playback. Please refer to lingo 0x04 commands 0x0002 through 0x000B for details. The following Extended Interface commands have been modified to support audiobook playback:
	SelectDBRecord
	GetNumberCategorizedDBRecords
	ReturnNumberCategorizedDBRecords
	RetrieveCategorizedDatabaseRecords
	ReturnCategorizedDatabaseRecord
	PlayStatusChangeNotification
	PlayControl
	SelectSortDBRecord. The sort order field is ignored for the case of an audiobook, as audiobooks are automatically sorted by name.
<b>Version 1.05</b>	This version adds several new lingo 0x00 commands and minor fixes to version 1.04. The changes include:
	Fixed a problem where iPod Accessory Protocol commands on the 30-pin connector did not wake up unpowered, sleeping iPods.
	Fixed the SelectDBRecord routine to play tracks from a stopped play state.
	Fixed RetrieveCategorizedDBRecords to retrieve all records from the start index when passed a count of -1.
	Fixed long record names resulting in large packets causing the iPod to reboot.
	Added missing begin and end FF or REW notification messages.
<b>Version 1.04</b>	This version adds some minor fixes to Version 1.03. The changes include:
	Fixed a problem where devices were intermittently not detected when the 30-pin connector was plugged in to the iPod.
	Optimized the iPod ReturnPlayStatus response to the GetPlayStatus message from the device. Command response is much faster.
<b>Version 1.03</b>	This version is the functional equivalent of protocol version 1.02 (1.03 is equal to 1.02). It is only reported on the iPod mini in software version 1.1. All of the changes reported for version 1.02 are applicable for 1.03 and no more.
<b>Version 1.02</b>	This version was released in the third generation (3G) iPod system software 2.2. The changes include:

Version	Changes
	Fixed a problem where the user's On-The-Go playlist was erased when the Extended Interface serial protocol was initialized.
	Fixed a problem where a playing iPod was stopped and playing state lost when the Extended Interface mode was initialized.
	Fixed a problem where the strings returned from the iPod were truncated if there were multibyte characters present.
	Fixed a problem where an iPod playing a track in Extended Interface mode would continue playing the same track, unpaused, when it was disconnected and reverted to the iPod UI mode.
	Fixed a problem where an iPod in Extended Interface mode failed to go into Sleep mode when power was removed from it.
	Fixed a problem where an iPod with a set alarm would honor the alarm while in Extended Interface mode or soon after disconnection.
	Fixed a problem where the pause indicator was shown when playback was stopped during Extended Interface mode.
<b>Version 1.01</b>	This version was released in the iPod mini version 1.0 system software with bug fixes and protocol improvements during February 2004. The changes include:
	Added the commands: <code>SelectSortDBRecord</code> , <code>SetCurrentPlayingTrack</code> , <code>GetNumPlayingTracks</code> , <code>GetMonoDisplayImageLimits</code> and <code>SetDisplayImage</code> .
	Changed the default sort order of returned or listed database items to match that of the iPod UI.
	Fixed a transmit problem where large packets being sent by the iPod would pause, mid-packet, until the beginning of another packet was received by the iPod.
	Fixed a problem where the iPod play/pause button remained active when the Extended Interface protocol was in control of the iPod.
<b>Version 1.0</b>	This is the base protocol released in October 2003 in system software version 2.1 of the 3G iPod.



# Accessory Identification

When it is connected to an iPod or iPhone, an accessory must identify itself using the Identify Device Preferences and Settings (IDPS) process described in this appendix.

**IMPORTANT:** The IDPS process is required for new iPod and iPhone accessory designs, to ensure their compatibility with future firmware. Existing accessory designs may continue to send the iAP General lingo command 0x13, `IdentifyDeviceLingoes`, identifying the lingoes they support and requesting immediate authentication. Note that IDPS requires accessory Authentication 2.0, as specified in ["Authentication"](#) (page 52), and the use of transaction IDs throughout each communication session, as specified in ["Transaction IDs"](#) (page 451).

## Using IDPS

The IDPS process begins when an accessory sends a `StartIDPS` command to an iPod (see ["Command 0x38: StartIDPS"](#) (page 108)). If the two are newly connected, it must be the first command the accessory sends. If the iPod refuses `StartIDPS` by returning a General lingo `ACK` command with a status of 0x04 (Bad Parameter), the accessory must assume it is connected to an iPod that doesn't support the IDPS process. In this case, the accessory must send the iPod an `IdentifyDeviceLingoes` command within 800 ms, requesting authentication.

**IMPORTANT:** If the accessory has access to Accessory Power through the 30-pin connector, it must always monitor that output from the iPod (pin 13). If Accessory Power goes low, even momentarily, the accessory must restart the IDPS process after it goes high. After Accessory Power goes high the accessory must wait at least 80 ms before starting the IDPS process or sending any other iAP commands.

During the IDPS process, the iPod accepts only the following General lingo commands from the accessory:

- 0x4B, `GetiPodOptionsForLingo` to determine the options the iPod supports for a specific lingo
- 0x39, `SetFIDTokenValues` to identify the accessory to the iPod
- 0x3B, `EndIDPS` to end the IDPS process

**IMPORTANT:** After an iPod or iPhone has successfully acknowledged an accessory's `StartIDPS` command, all subsequent iAP command packets must include transaction IDs, regardless of lingo, as specified in ["Transaction IDs"](#) (page 451).

The IDPS process lets the iPod receive preference information from the accessory during the initial handshake sequence. Unlike the process using `IdentifyDeviceLingoes`, the audio/video routing and other iPod settings are established before authentication begins. Once the IDPS process begins, the iPod will not enable

preferences such as line-out and video-out by default; therefore it is up to the accessory to explicitly enable all desired preferences. During the IDPS process, the iPod identifies connected accessories as uniquely as possible.

## IDPS Commands

The General lingo IDPS commands that support accessory identification are listed below and documented in detail in the sections that follow. All commands are protocol version 1.09 and require authentication 2.0.

**Table A-1** IDPS General lingo commands

CmdID	Name	Direction	Payload:bytes
0x38	StartIDPS	Dev to iPod	{transID;2}
0x39	SetFIDTokenValues	Dev to iPod	{transID;2, numFIDTokenValues;1, FIDTokenValues;<var>}
0x3A	RetFIDTokenValueACKs	iPod to Dev	{transID;2, numFIDTokenValueACKs;1, FIDTokenValueACKs;<var>}
0x3B	EndIDPS	Dev to iPod	{transID;2, accIDPSStatus;1}
0x3C	IDPSStatus	iPod to Dev	{transID;2, status;1}

**Note:** The `transID` parameters in the foregoing table are described in ["Transaction IDs"](#) (page 451).

For details of these commands, see ["The Protocol Core and the General Lingo"](#) (page 47).

## Sample IDPS Command Sequences

This section lists two typical command sequences using IDPS:

- [Table A-2](#) (page 446) illustrates the basic IDPS command sequence.
- [Table A-3](#) (page 447) illustrates an IDPS command sequence followed by authentication and accessory support for Digital Audio (Lingo 0x0A).

**Table A-2** IDPS process up to authentication

Step	Accessory command	iPod command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'

Step	Accessory command	iPod command	Comment
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/2/3/4 options 0x00000002 device ID 0x00000200); AccCapsToken = 0x0000000000000205; AccInfoToken = Acc name (Test-Accessory); AccInfoToken = Acc FW version (v1.1.1); AccInfoToken = Acc HW version (v2.2.2); AccInfoToken = Acc manufacturer (Apple; Inc.); AccInfoToken = Acc model number (Test-Model); SDKProtocolToken = 1 (com.Apple.ProtocolMain); SDKProtocolToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; SDKProtocolToken = (1) accepted; SDKProtocolToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication-Info	no params

**Table A-3** IDPS process with authentication and Digital Audio lingo

Step	Accessory command	iPod command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'

Step	Accessory command	iPod command	Comment
3	SetFIDTokenValues		setting 11 FID tokens ; IdentifyToken = (lingoes: 0/2/3/4/10 options 0x00000002 device ID 0x00000200); AccCapsToken = 0x0000000000000215; AccInfoToken = Acc name (Test-Accessory); AccInfoToken = Acc FW version (v1.1.1); AccInfoToken = Acc HW version (v2.2.2); AccInfoToken = Acc manufacturer (Apple; Inc.); AccInfoToken = Acc model number (Test-Model); SDKProtocolToken = 1 (com.Apple.ProtocolMain); SDKProtocolToken = 2 (com.Apple.ProtocolAlt); iPodPreferenceToken = (setting 'off' for preference class 'video out setting' with restore on exit 'not selected'); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'not selected')
4		RetFIDTokenValueACKs	11 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; SDKProtocolToken = (1) accepted; SDKProtocolToken = (2) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (line out usage) accepted
5	EndIDPS		status 'finished with IDPS; proceed to authentication'
6		IDPSStatus	status 'ready for auth'
7		GetDevAuthentication-Info	no params
8	RetDevAuthentication-Info		returning version of authentication 2.0 and X.509 certificate; see <a href="#">Table 2-37</a> (page 85)
9		AckDevAuthentication-Info	acknowledging 'auth info supported'
10		GetAccSampleRateCaps	no params
11		GetDevAuthentication-Signature	offering challenge for the accessory to sign and return; see <a href="#">Table 2-39</a> (page 87)

Step	Accessory command	iPod command	Comment
12	RetAccSampleRateCaps		returning sample rates '8000 11025 12000 16000 22050  24000 32000 44100 48000'
13	RetDevAuthentication-Signature		returning digital signature; see <a href="#">Table 2-40</a> (page 88)
14		NewiPodTrackInfo	sample rate 44100; Sound Check value 0; track volume adjustment 0
15	AccAck		acknowledging 'NewiPodTrackInfo'
16		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
17		NewiPodTrackInfo	sample rate 44100; Sound Check value 0; track volume adjustment 0
18	AccAck		acknowledging 'NewiPodTrackInfo'
19		NewiPodTrackInfo	sample rate 44100; Sound Check value 0; track volume adjustment 0
20	AccAck		acknowledging 'NewiPodTrackInfo'

## iPod Event Notifications

Notifications of one type, Flow Control, are enabled by default if the accessory uses IDPS. Accessories that use `IdentifyDeviceLingoes` must enable Flow Control notifications explicitly. Receiving Flow Control notifications lets accessories cope with situations where the iPod's incoming queue is full and it is unable to accept more packets.

To work reliably with all past and future iPods, an accessory should first send a `GetSupportedEventNotification` command to check which notification types the iPod supports. The iPod can respond in one of two ways:

- The iPod can return an ACK command with a Bad Parameter error. This means that the iPod does not support the `GetSupportedEventNotification` command. The accessory must then send a `SetEventNotification` command that sets only the `FlowControl` bit. The iPod may respond in one of two ways:
  - If the iPod responds with a successful ACK command, it means that the iPod supports Flow Control notifications and the accessory should be prepared to handle them.
  - If the iPod responds with an ACK command that passes a Bad Parameter error, it means that iPod does not support notifications.
- The iPod can return a `RetSupportedEventNotification` command with a notification bitmask indicating which notifications it supports. The accessory must then send a `SetEventNotification` command, passing the `FlowControl` bit plus any other bits chosen from the returned bitmask.



# Transaction IDs

---

Once an accessory has started the IDPS process, beginning with and including the `StartIDPS` command, the accessory must include 16-bit transaction ID parameters in every iAP packet. This appendix specifies the rules that must be followed when creating or interpreting the values of these parameters.

## Using Transaction IDs

The purpose of transaction IDs is to uniquely associate iAP commands with their responses, regardless of the order in which commands and their responses may be transmitted.

**Note:** Transaction ID fields are part of the packet payload and must be counted when specifying the data lengths of iAP commands.

## Generating and Returning Transaction IDs

---

An attached accessory's responsibility for handling transaction IDs can be summarized by the following rules:

- Every time the accessory responds to a command from the iPod or iPhone that contains a transaction ID, it must include that ID in its response.
- To generate transaction IDs for the commands it sends, the accessory must initialize a 16-bit counter to 0x0000 every time it is connected to an iPod or iPhone. It must then use the counter value as the transaction ID for every command it sends and increment the counter afterward. The counter may roll over to 0x0000 after it reaches 0xFFFF.
- If the accessory receives no response to a command that it has sent, it may send another command with the same payload, but the transaction ID must be different.
- The accessory must ignore any response from the iPod or iPhone whose transaction ID does not match that of a previous command it has sent.
- The iPod or iPhone will ignore any response from the accessory whose transaction ID does not match that of a previous command it has sent.

**Note:** The iPod or iPhone does not increment transaction ID values during authentication. The Accessory must continue to respond to iPod commands with the transaction ID included in the iPod's command.

## Enabling and Disabling Transaction ID Support

---

Accessories must enable their support for transaction IDs according to the following rules:

- Support for transaction IDs must be enabled upon the rising edge of power from the iPod or iPhone on pin 13 (Accessory Power) of the 30-pin connector (see *iPod/iPhone Hardware Specifications*).
- Support for transaction IDs must be enabled before the accessory sends a `StartIDPS` command.

The accessory must continue to enable its transaction ID support for all iAP commands until it encounters one of the following situations:

- Support for transaction IDs must be disabled upon receipt of a General lingo `ACK` command without a transaction ID. Such commands have a payload length value (byte 2) of either 0x04 or 0x08.
- Support for transaction IDs must be disabled upon receipt of a `RequestIdentify` command.
- Support for transaction IDs must be disabled before sending an `IdentifyDeviceLingoes` command.

**Note:** When responding to an accessory's command containing a transaction ID, an iPod with older firmware that does not support transaction IDs will send an `ACK` command with a Bad Parameter status (0x04) and no transaction ID.

## iPod Acknowledgment of Transaction ID Commands

To support commands with transaction IDs, the General lingo `ACK` command has been expanded to four formats, listed in [Table B-1](#) (page 452). The new command details are presented in "[Command 0x02: General Lingo ACK](#)" (page 452).

**Table B-1** Forms of the General lingo `ACK` command

Transaction ID	Command pending	Payload length	Authentication required
Yes	No	0x06	Authentication 2.0B
Yes	Yes	0x0A	Authentication 2.0B
No	Yes	0x08	None
No	No	0x04	None

## Command 0x02: General Lingo ACK

Direction: iPod to Device

The iPod sends a General lingo `ACK` command to an attached accessory to acknowledge receipt of a General lingo command and advise it of the command's completion status and errors. The General lingo `ACK` command has these forms:

- If the command being acknowledged sent a transaction ID, the `ACK` packet has the format shown in [Table B-2](#) (page 453). Accessories must manage transaction IDs as specified in "[Transaction IDs](#)" (page 451).



- If the acknowledgment status value being returned is 0x06 (Command Pending), the ACK packet has the format shown in [Table B-4](#) (page 454) (or [Table B-5](#) (page 455) if acknowledging a command without a transaction ID).
- If none of the foregoing is the case, the ACK packet has the format shown in [Table B-6](#) (page 455).

It is the responsibility of the accessory developer to recognize which ACK format will be returned for each General lingo command sent by the accessory and to parse it correctly.

**Table B-2** General lingo ACK packet with transaction ID

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command ID: ACK
5	0xNN	transID [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451)
6	0xNN	transID [bits 7:0]
7	0xNN	ackStatus: Status of command received. See <a href="#">Table B-3</a> (page 453).
8	0xNN	origCmdID: ID of command received.
9	0xNN	Checksum

**Table B-3** ACK command error codes

Value	Description
0x00	Success (OK)
0x01	ERROR: Unknown database category
0x02	ERROR: Command failed
0x03	ERROR: Out of resources
0x04	ERROR: Bad parameter
0x05	ERROR: Unknown ID
0x06	Command Pending; see <a href="#">Table B-5</a> (page 455)
0x07	ERROR: Not authenticated
0x08	ERROR: Bad authentication version

Value	Description
0x09	ERROR: Accessory power mode request failed
0x0A	ERROR: Certificate invalid
0x0B	ERROR: Certificate permissions invalid
0x0C	ERROR: File is in use
0x0D	ERROR: Invalid file handle
0x0F	ERROR: Operation timed out
0x10	ERROR: Command unavailable in this iPod mode
0x11	ERROR: Invalid accessory resistor ID value
0x12	ERROR: Accessory not grounded
0x13	Multisection data section received successfully; see <a href="#">"Multisection Data Transfers"</a> (page 459)
0x14–0xFF	Reserved

If the status returned by the ACK command is Command Pending, an additional field is added to the ACK packet that represents the amount of time, in milliseconds, that an accessory must wait to receive the final packet indicating that the current command completed or returned one of the error codes listed in [Table B-3](#) (page 453).

After receiving a Command Pending ACK, the device must wait for up to the specified number of milliseconds for a final ACK response. If no final ACK packet is received before the specified amount of time expires, the device should retry the command.

**Table B-4** General lingo ACK packet with transaction ID and Command Pending status

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x0A	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command: ACK
5	0xNN	transID [bits 15:8]: Transaction ID; see <a href="#">"Transaction IDs"</a> (page 451)
6	0xNN	transID [bits 7:0]
7	0x06	Command result status: Command Pending
8	0xNN	The ID of the command being acknowledged.

Byte number	Value	Comment
9	0xNN	Maximum amount of time to wait for pending response, in milliseconds (bits 31:24).
10	0xNN	Maximum pending wait, in milliseconds (bits 23:16)
11	0xNN	Maximum pending wait, in milliseconds (bits 15:8)
12	0xNN	Maximum pending wait, in milliseconds (bits 7:0)
13	0xNN	Checksum

**Table B-5** General lingo ACK packet with Command Pending status

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command: ACK
5	0x06	Command result status: Command Pending
6	0xNN	The ID of the command being acknowledged.
7	0xNN	Maximum amount of time to wait for pending response, in milliseconds (bits 31:24).
8	0xNN	Maximum pending wait, in milliseconds (bits 23:16)
9	0xNN	Maximum pending wait, in milliseconds (bits 15:8)
10	0xNN	Maximum pending wait, in milliseconds (bits 7:0)
11	0xNN	Checksum

**Table B-6** Default General lingo ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x02	Command: ACK

Byte number	Value	Comment
5	0xNN	Command result status. Possible values are shown in <a href="#">Table B-3</a> (page 453).
6	0xNN	The ID of the command being acknowledged
7	0xNN	Checksum

## Examples of Transaction IDs

[Table B-7](#) (page 456) through [Table B-10](#) (page 458) illustrate the use of transaction IDs in some typical iAP command flows.

**Table B-7** Example of IDPS and authentication

Step	Accessory command	iPod command	Comment
1	StartIDPS		Transaction ID = 0x0001 (accessory counter starts counting).
2		ACK	Transaction ID = 0x0001; response to Step 1
3	SetFIDTokenValues		Transaction ID = 0x0002.
4		RetFIDTokenValueACKs	Transaction ID = 0x0002; assume some token value was not acknowledged.
5	SetFIDTokenValues		Transaction ID = 0x0003; retry command with a different transaction ID
6		RetFIDTokenValueACKs	Transaction ID = 0x0003; response to Step 5.
7	EndIDPS		Transaction ID = 0x0004.
8		IDPSStatus	Transaction ID = 0x0004; response to Step 7
9		GetDevAuthentication-Info	Transaction ID = 0x0001; first command initiated by the iPod (iPod counter starts counting).
10	RetDevAuthentication-Info		Transaction ID = 0x0001; response to Step 9.
11		AckDevAuthentication-Info	Transaction ID = 0x0001; response to Step 10.

**Table B-8** Example of entering Remote UI mode

Step	Accessory command	iPod command	Comment
1	StartIDPS		Transaction ID = 0x0001 (accessory counter starts counting).
2		ACK	Transaction ID = 0x0001; response to Step 1
3	SetFIDTokenValues		Transaction ID = 0x0002.
4		RetFIDTokenValueACKs	Transaction ID = 0x0002; response to Step 3.
5	EndIDPS		Transaction ID = 0x0003.
6		IDPSStatus	Transaction ID = 0x0003; response to Step 5.
Enter background authentication state; see "Authentication" (page 52).			
7	EnterRemoteUIMode		Transaction ID = 0x0004.
8		ACK	Transaction ID = 0x0004; response to Step 7: command pending.
9		ACK	Transaction ID = 0x0004; sent after the iPod has entered Remote UI mode. Note that the same transaction ID value is used.

**Table B-9** Example of getting track artwork data

Step	Accessory command	iPod command	Comment
1	StartIDPS		Transaction ID = 0x0001 (accessory counter starts counting).
2		ACK	Transaction ID = 0x0001; response to Step 1
3	SetFIDTokenValues		Transaction ID = 0x0002.
4		RetFIDTokenValueACKs	Transaction ID = 0x0002; response to Step 3.
5	EndIDPS		Transaction ID = 0x0003.
6		IDPSStatus	Transaction ID = 0x0003; response to Step 5.
Enter background authentication state; see "Authentication" (page 52).			
7	GetTrackArtworkData		Transaction ID = 0x0004.

Step	Accessory command	iPod command	Comment
8		RetTrackArtworkData	Transaction ID = 0x0004; Response is divided into multiple packets with the same transaction ID.
9		RetTrackArtworkData	Transaction ID = 0x0004.
10		RetTrackArtworkData	Transaction ID = 0x0004; Last packet returning artwork data.

**Table B-10** Example of sending notifications

Step	Accessory command	iPod command	Comment
1	StartIDPS		Transaction ID = 0x0001 (accessory counter starts counting).
2		ACK	Transaction ID = 0x0001; response to Step 1
3	SetFIDTokenValues		Transaction ID = 0x0002.
4		RetFIDTokenValueACKs	Transaction ID = 0x0002; response to Step 3.
5	EndIDPS		Transaction ID = 0x0003.
6		IDPSStatus	Transaction ID = 0x0003; response to Step 5.
7		iPodNotification	Transaction ID = 0x0001; first command initiated by the iPod (iPod counter starts counting).
8		iPodNotification	Transaction ID = 0x0002; second command initiated by the iPod.
9	RdsReadyNotify		Transaction ID = 0x0004; next accessory transaction ID.
10	RdsReadyNotify		Transaction ID = 0x0005.

# Multisection Data Transfers

---

Three Location lingo commands may need to transfer amounts of data larger than the data receiver's maximum incoming packet size: `RetDevData`, `SetDevData`, and `AsyncDevData`. This appendix specifies how large amounts of data are transferred in sections, using multiple packets of the same command.

## Multisection Transfer Process

The packets of commands that can transfer data in multiple sections contain two 16-bit parameters that uniquely identify transferred data sections:

- Parameter `sectCur` is the index number of the data section in that packet. The first packet has index 0x0000.
- Parameter `sectMax` is the index number of the last data section

When a data transfer fits into a single packet, both `sectCur` and `sectMax` must be set to 0x0000. When a command transfers data in multiple sections, the transaction ID of the command that requested the transfer must be used for all response sections and the `sectCur` value of each serial packet must increment by 0x0001. For optimal data throughput, data sections must be the maximum size permitted by the receiver, less any command header overhead. The last data section may be smaller than the maximum size if the total data transfer size is not a multiple of the section size.

An accessory may either send or receive multisection data. When the accessory sends multisection data to an iPod, using `RetDevData` or `AsyncDevData`, it must wait after each packet for the iPod to reply with an `iPodACK` command, as detailed in [Multisection iPodACK Command](#) (page 460). When the iPod sends multisection data to an accessory, using `SetDevData`, the accessory must reply with a `DevACK` command after each packet, as detailed in [Multisection DevACK Command](#) (page 460).

Regardless of whether the accessory is sender or receiver, the acknowledgment command must have the same transaction ID as the data transfer command. When acknowledging any packet before the last one, the acknowledgment must also return the packet's `sectCur` value plus an `ackStatus` of Section Received OK (0x13) if the section was transferred successfully. When the last section has been received successfully, the receiver must send an `ackStatus` of Command OK (0x00), without a `sectCur` value, to indicate that the entire data transfer has finished.

If an error occurs during a multisection transfer, the receiver must send an acknowledgment with a nonzero `ackStatus` other than 0x13. This means the transfer has failed. The sender must stop sending data sections with the same transaction ID. If the receiver does not acknowledge a multisection packet within 400 ms, the transfer has also failed; the sender must send the receiver an acknowledgment with the transfer's transaction ID and a Command Timeout (0x0F) status. When a transfer fails, the sender can try to send the same data again, starting from the beginning, with a different transaction ID.

## Multisection iPodACK Command

Direction: iPod to Accessory

An iPodACK command is sent by the iPod in response to each multisection data transfer packet sent by RetDevData or AsyncDevData. The iPod may send either of two forms of the iPodACK packet:

- When any section before the last section has been received successfully, the iPod sends the packet shown in [Table C-1](#) (page 460). The accessory may send a new packet immediately.
- When the last section has been received successfully, the iPod sends the packet shown in [Table 3-270](#) (page 333) with `ackStatus = 0x00`, indicating that the multisection data transfer is complete. See [Command 0x80: iPodACK](#) (page 333) for details of this form of the iPodACK command. The iPod also sends an acknowledgment of this form, with a nonzero `ackStatus`, when an error has occurred in receiving any packet.

**Table C-1** Multisection iPodACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x80	Command ID: iPodACK
5	0xNN	transID [bits 15:8]: Transaction ID of command being acknowledged; see <a href="#">"Transaction IDs"</a> (page 451)
6	0xNN	transID [bits 7:0]
7	0x13	ackStatus: Section received successfully
8	0xNN	cmdIDorig: Original command ID for which this response is being sent.
9	0xNN	sectCur [Bits 15:8]: Section index for which this response is being sent.
10	0xNN	sectCur [Bits 7:0]
11	0xNN	Checksum

## Multisection DevACK Command

Direction: Accessory to iPod

The accessory must send a DevACK command in response to each multisection data transfer packet sent by the iPod via SetDevData. The accessory may send either of two forms of the DevACK packet:



- When it has successfully received any section before the last section, the accessory must send the packet shown in [Table C-2](#) (page 461). The iPod will send a new packet immediately.
- When it has successfully received the last section, the accessory must send the packet shown in [Table 3-247](#) (page 317) with `ackStatus = 0x00`, indicating that the multisection data transfer is complete. See [Command 0x00: DevACK](#) (page 317) for details of this form of the `DevACK` command. The accessory must also send an acknowledgment of this form, with a nonzero `ackStatus`, whenever it has detected an error in receiving any packet.

**Table C-2** Multisection DevACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x08	Length of packet payload
3	0x0E	Lingo ID: Location lingo
4	0x00	Command ID: DevACK
5	0xNN	<code>transID</code> [bits 15:8]: Transaction ID of command being acknowledged; see <a href="#">"Transaction IDs"</a> (page 451)
6	0xNN	<code>transID</code> [bits 7:0]
7	0x13	<code>ackStatus</code> : Section received successfully
8	0x08	<code>cmdIDOrig</code> : ID of command ( <code>SetDevData</code> ) for which this response is being sent.
9	0xNN	<code>sectCur</code> [Bits 15:8]: Section index for which this response is being sent.
10	0xNN	<code>sectCur</code> [Bits 7:0]
11	0xNN	Checksum



# iTunes Tagging

---

This appendix explains the iTunes tagging feature and specifies the requirements for an HD or FM radio accessory to support it.

## The iTunes Tagging Experience

To experience the capabilities of the iTunes tagging feature, an iPod user typically goes through the following steps:

1. While listening to broadcast music on an HD or FM radio accessory, the user hears a song and decides to tag it for future review or purchase by pressing a special button or other user interface element.
2. The radio accessory, equipped with a tag button or equivalent software action, stores metadata for the currently playing song. The user tags the song without needing to know anything about it or its originating station. The song information is stored in the radio accessory until an iPod is connected to it.
3. When an iPod is connected to it, the radio transfers the tagged song information to the iPod. The iPod stores the data until the iPod is synched with iTunes.
4. When the iPod is connected to the user's computer, iTunes imports the tag data, analyzes it, and presents it to the user in the form of a tagged playlist. iTunes displays the song title, artist, album and other information for each tagged song.
5. Using iTunes, the user may save, review or delete any item in the tagged playlist and may easily purchase any of the listed songs through the iTunes store.

## Tagging Feature Components

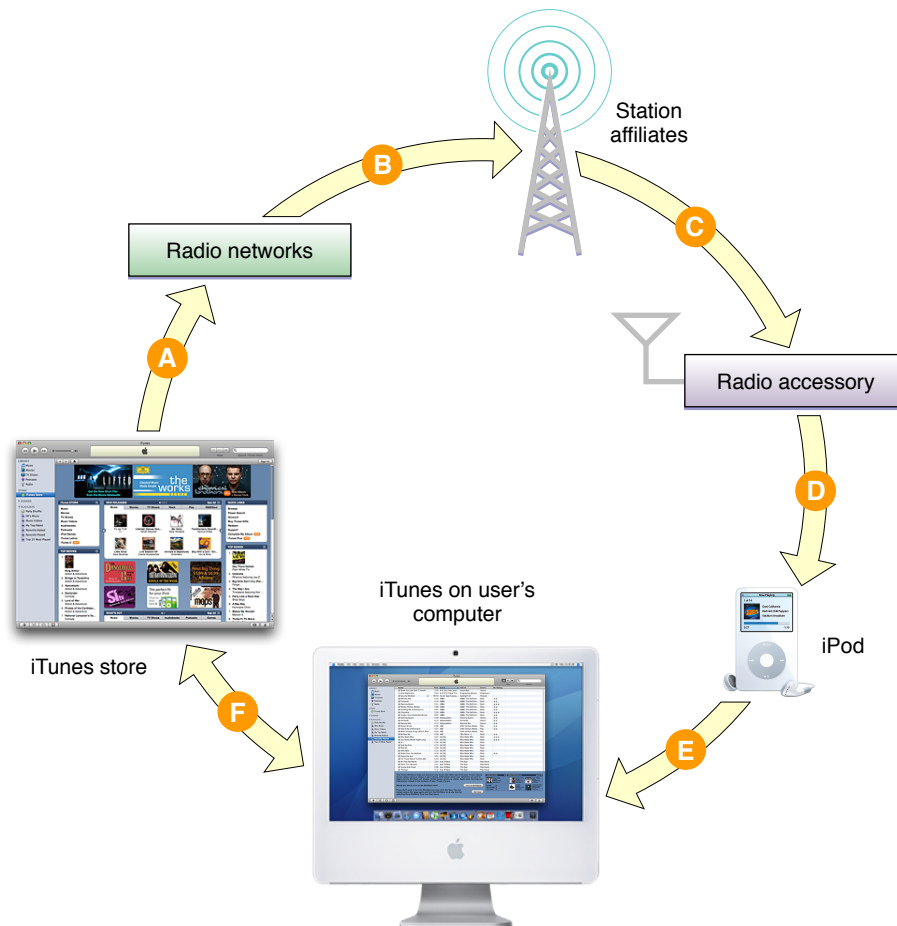
To implement the user experience described in the previous section, the iTunes tagging feature includes the following major components:

- The **iTunes store** creates and publishes unique identifiers for media available in its catalog, and allows users of accessories that support radio tagging for iTunes to purchase tagged songs.
- **Radio networks and station affiliates** receive iTunes store data and broadcast it along with program music and other content.
- **HD or FM radio accessories for the iPod** are designed to enable the tagging feature. These radio receivers may be portable, used in the home or in a car.
- An **iPod** is a media player used to transfer tags from iPod accessories to iTunes.

- The **iTunes application** is a desktop media player for managing media content, including newly-discovered music via the iTunes tagging feature.

The data flows among the components of the iTunes tagging feature listed above are diagrammed in [Figure D-1](#) (page 464).

**Figure D-1** iTunes tagging feature data flows



Data flow	Description
A	An enterprise partner feed from Apple contains iTunesSongIDs, track names, artists, and album names for the contents of the iTunes store.
B	Network data is sent to the station affiliates, including Apple-supplied data.
C	Affiliates broadcast metadata with each song, containing the song's iTunesSongID, track name, and artist name.
D	The radio accessory writes XML tag data to a file on an attached iPod, using the iAP Storage lingo; the iPod signs the file.

Data flow	Description
E	iTunes verifies and uploads the iPod's tag files and presents a tagged music playlist to the user.
F	The user clicks a "Buy" button next to the tagged music and downloads the music from the iTunes Store.

## Radio Accessory Requirements

Radio accessories that support the iTunes tagging feature described in this specification must perform the following actions:

- Authenticate themselves to the attached iPod. This requires that the accessory contain an authentication coprocessor supplied by Apple. For technical details, see Apple's *iPod Authentication Coprocessor 2.0B Specification*.
- Conform to the user interface requirements described in ["Accessory User Interface"](#) (page 482).
- Provide static storage capacity for at least 50 tags, and provide enough RAM to cache two tags in cases of tag ambiguity (see ["Resolving Tag Ambiguity"](#) (page 476)). Accessories should store tags in a data structure and only generate XML when writing files to the iPod. Tag data may be stored internally in any format convenient to the accessory.
- Generate tag data from the radio broadcast's metadata. Accessories supporting iTunes Tagging must populate every XML field for which metadata is received.
- Generate files from the tag data and write these files to the attached iPod. The files must be XML-formatted **plists**, as specified in ["Tag Data Writing Process"](#) (page 477). The plist fields in these files are specified in ["Data Transfer to the iPod"](#) (page 479), and the iAP commands used to write a file to the iPod are defined in ["Lingo 0x0C: Storage Lingo"](#) (page 300).

## HD Radio Tagging

In addition to their audio content, HD radio broadcasts contain Program Service Data (PSD) and Station Information Service (SIS) data. The HD radio accessory must capture these data streams and write their information to plist fields as shown in [Table D-8](#) (page 479). The accessory must populate every plist field for which it receives data from the radio broadcast, including program service data (PSD) and station information services (SIS).

**Note:** Mac OS X plist files are UTF-8 encoded; the PSD data sent by HD radio broadcasters is ISO-8859-1 encoded. The radio accessory must convert the ISO-8859-1 data to UTF-8 before writing the plist file to the iPod. If the plist file contains ISO-8859-1 characters in the range 0x80 to 0xFF, iTunes cannot parse the file properly and the tag will be lost. The XML markup delimiters `&`, `<`, and `>` must be written as `&amp;`, `&lt;`, and `&gt;`; —otherwise iTunes cannot open the file.

## FM Radio Tagging

FM radio uses the RDS/RBDS subcarrier system to broadcast data, in addition to its audio content. For information about RBDS, see the *United States RBDS Standard, NRSC-4-A*, available at [www.nrscstandards.org](http://www.nrscstandards.org). FM tagging data is broadcast inside two ODAs (Open Data Applications): a RadioText Plus (RT+) ODA and a private iTunes Tagging ODA.

The FM radio accessory must capture the data streams from both these ODAs and write their information to plist files, as described in "Capturing and Storing Tag Data" (page 477). During this process, the accessory must populate every plist field listed in Table D-8 (page 479) for which it receives metadata from the radio broadcast. For an FM tag to be usable, its plist entry must provide valid data for at least the `Title` and `Artist` fields.

### The RT+ ODA

RadioText Plus (RT+) is a technology that tags RDS RadioText (RT) messages so specific content can be retrieved from them. RT+ tags are transmitted in RDS broadcasts as an ODA with an Application ID (AID) of 0x4BD7. You can download the RT+ specification at [www.rds.org.uk/rds98/pdf/R06\\_040\\_1.pdf](http://www.rds.org.uk/rds98/pdf/R06_040_1.pdf).

Each RT+ tag contains three elements:

- The RT content type
- The position inside the RT message of the first character of content to be retrieved
- The length of the content to be retrieved.

The RT+ content types used for iTunes FM radio tagging are listed in Table D-1 (page 466). The corresponding plist fields sent to the iPod are described in Table D-8 (page 479).

**Table D-1** RT+ content types

RT+ type	iTunes plist field
ITEM.TITLE	Name
ITEM.ARTIST	Artist
ITEM.ALBUM	Album
ITEM.GENRE	Genre
STATIONNAME.SHORT, STATIONNAME.LONG	StationCallLetters
INFO.URL	PodcastFeedURL

RT+ type	iTunes plist field
PROGRAMME.HOMEPAGE	StationURL
PROGRAMME.SUBCHANNEL	ProgramNumber

## The iTunes Tagging ODA

---

The iTunes ODA used for FM broadcasts encodes tag information as **elements**. Currently 6 element types are defined and another 10 are reserved for future use, as shown in [Table D-2](#) (page 467).

**Table D-2** FM tag element types

ID type	Description
0x0	Event Control
0x1	iTunes Song ID
0x2	iTunes Artist ID
0x3	Reserved
0x4	Industry ID
0x5	iTunes Storefront
0x6	Extended ID
0x7-0xF	Reserved

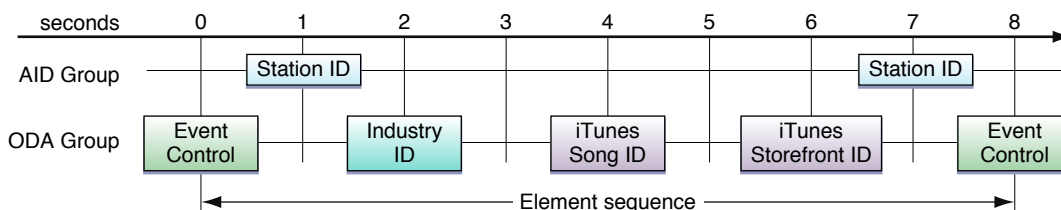
All currently defined FM tag elements carry 32 bits of data and are transmitted using a single ODA application group. If necessary, future versions of the iTunes FM tagging protocol may accommodate larger data types by chaining several ODA groups together.

FM tag elements are normally encrypted before transmission. Because the transmission must be decrypted by the receiver, it is keyed to a composite of values contained in the RDS data. This process is described in ["FM Tag Decryption"](#) (page 471).

A complete iTunes tagging transmission normally consists of several element types in an **element sequence**. The elements in the sequence are transmitted at a relatively low rate (about one element every 2 seconds), so the RDS bandwidth consumed by iTunes tagging is relatively small. When a complete sequence of elements has been transmitted the process repeats itself. [Figure D-2](#) (page 468) shows the transmission timeline of a typical element sequence. In this example, the sequence consists of 4 element types and a complete transmission of the sequence occurs every 8 seconds.

**Note:** To assure that tag data is always available, the maximum recommended time between transmissions is 10 seconds.

**Figure D-2** iTunes tagging element sequence



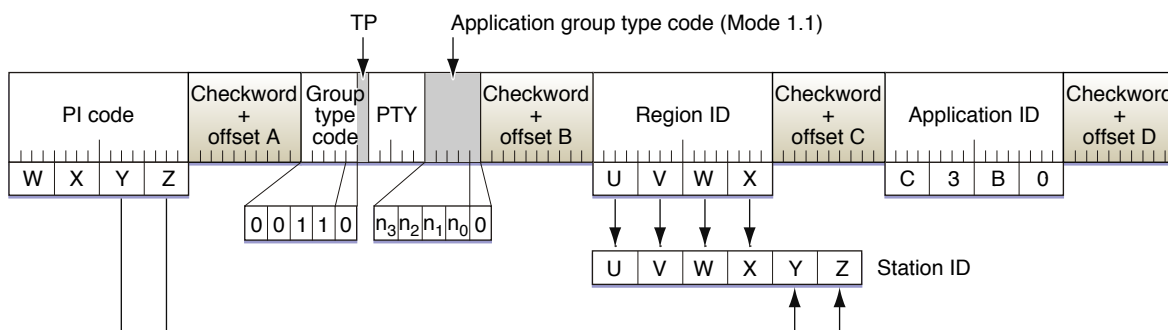
### Use of the RDS Group Type 3A

Because the RT+ and FM iTunes tagging data are contained in ODAs, they use the RDS group type 3A, also called the AID (Application ID) Group type. This RDS group type includes 16 message bits for use by each ODA (block C) and indicates the application group type (if any) used for transmission of additional ODA data. Tag data is transmitted in mode 1.1, meaning that the broadcaster allocates an available type A group for transmission of iTunes tag element data. RDS 3A group messages with Application ID 0xC3B0 indicate which RDS group type will be used to carry the iTunes tagging information. The accessory must parse group type 3A messages and save the application group type code (n3-n0), because this indicates the ODA group type on which tag data will be broadcast.

**Note:** The accessory must parse and save the application group type code for each iTunes-specific type 3A message, because the type code for subsequent iTunes data may change during the broadcast.

The recommended transmission rate for the iTunes tag 3A group is once every 7 seconds, with a maximum repeat delay of once every 10 seconds. Upon tuning to a new frequency, a tagging-enabled FM receiver waits for the reception of a 3A group containing the assigned Application ID number; this indicates that the broadcast includes iTunes tag data. The RDS 3A group structure for iTunes tagging is shown in [Figure D-3](#) (page 468).

**Figure D-3** RDS 3A group for iTunes tagging



The Region ID code shown in [Figure D-3](#) (page 468) combines the extended country code (see Annex N of the RBDS specification) and the upper byte of the PI (Program Identification) code: the upper byte "UV" of the Region ID carries the extended country code and the lower byte "WX" mirrors the upper byte of the PI



code. The receiver must combine the 16-bit Region ID with the lower byte of the PI code to form a 24-bit Station ID, a guaranteed unique identifier of a particular radio station. The 24-bit Station ID has the byte order UVWXYZ, as shown in the diagram.

**Note:** It is possible that the upper byte of the PI code may not exactly mirror the lower byte of the Region ID. For this reason, a receiver must construct the Station ID exactly as specified above. Always use the bytes transmitted in Region ID; ignore the upper byte of the PI code.

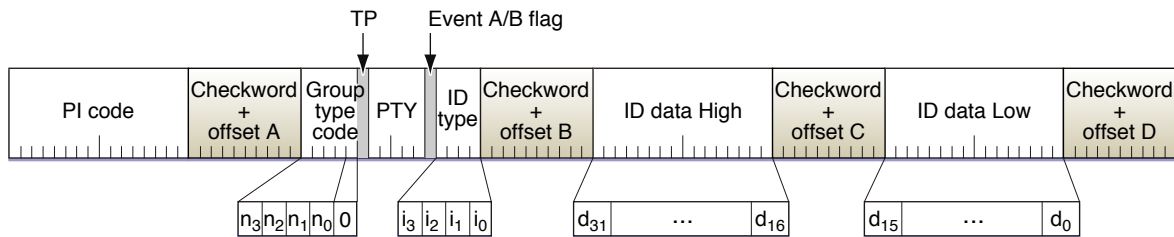
The Station ID is intended for use in affiliate programs. Because affiliate programs normally involve an online merchant, that merchant or an appropriate third party maintains a table translating the Station ID into a more traditional Affiliate ID. For example, a range of Station ID codes may map to a single group identifier for the purposes of an affiliate program.

## Tagging Application Group

The iTunes tag element data is transmitted using an RDS type A application group (also called an ODA Group), as shown in [Figure D-4](#) (page 469). The Group Type Code (n3-n0) of this message should correspond to that broadcast as the application group type code, which was part of the Group 3A message with AID 0xC3B0.

[Figure D-4](#) (page 469) depicts the data before encryption at the transmitter and following successful decryption at the receiver. The encryption process scrambles the ID Type (bits i3 to i0) as well as the ID data (bits d31 to d16 and d15 to d0). The remaining data (PI code, group type code, TP flag, PTY and Event A/B flag) are always transmitted in the clear.

**Figure D-4** RDS application group for iTunes tagging



The ID Type indicates the specific type of element data carried by an instance of the ODA group. There are 16 possible element types of which 6 are currently defined (see [Table D-2](#) (page 467)). The unassigned elements are available for future use. Types 00000 and 1111 are reserved as special-purpose control elements; all other types are available for content identification codes. Data received with an unassigned element type must be written to an `UnknownData` field in the plist sent to the iPod; see ["Handling Unknown Metadata Types"](#) (page 475).

The ID data field carries the actual data associated with an element. For all currently-defined numeric identifiers (iTunes Song, iTunes Artist, iTunes Storefront, Extended ID and Industry ID) these bits are interpreted as 32-bit unsigned integer values. The Event Control element uses an alternate definition of the data bits, as detailed in ["FM Tagging Event Control"](#) (page 470). The Event A/B flag is a toggle bit that changes state each time the broadcast content changes. This tells the receiver that it should clear any temporary buffers associated with element data in preparation for reception of a new set of data. In a typical music broadcast, the Event A/B flag toggles at the transition from one song to the next, at the transition from music to non-music content (advertisement break) and at the transition from non-music content back to music. Even if there are no

elements associated with a particular piece of content, the A/B flag still toggles at the start of that content to mark the end of previously identified content. In such a situation, an element sequence consisting only of the Event Control element is transmitted as a carrier for the A/B flag.

## Normal Mode and Test Mode

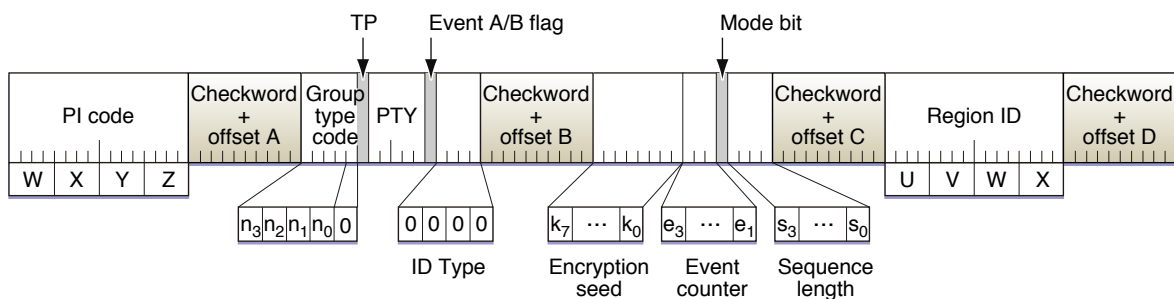
Two modes of operation are defined for the RDS transmission of iTunes tag data: **normal mode** and **test mode**. The Region ID section of Group Type Code 3A messages with AID 0xC3B0 indicate what mode is being used for broadcast. In normal mode the message bits of the AID group carry a 16-bit Region ID and encryption/decryption is enabled. In test mode the message bits are set to zero (0x0000) and encryption/decryption is disabled. Tagging-enabled FM receivers must support both operating modes. In test mode, the Station ID is constructed using Region ID and the PI code lower byte as carried by the Event Control element (See ["FM Tagging Event Control"](#) (page 470)). In test mode it is not possible to construct a Station ID from the 3A group because the Region ID is set to zero.

## FM Tagging Event Control

**Event Control** is a special-purpose element used to indicate a change in the content being broadcast as well as to indicate the overall status of the iTunes tag transmission.

The **event counter** is a 4-bit count that is incremented each time the on-air content changes. After the count reaches 15 it wraps back to 0. The 3 most significant bits of the count (bits e3-e1) are carried in the Event Control element, shown in [Figure D-5](#) (page 470). The least significant bit of the counter (bit e0) is carried in every element type and is called the Event A/B Flag.

**Figure D-5** Event Control element



When set to 1, the mode bit shown in [Figure D-5](#) (page 470) indicates that the event currently being broadcast has been successfully mapped to at least one type of content identifier and that identifiers are being transmitted as part of the element sequence. If no mapping exists for the current event, the mode bit is set to 0. The Sequence length bits (bits s<sub>3</sub>-s<sub>0</sub>) indicate the number of element types included in the element sequence for the current event. A value of 0000 indicates that the sequence consists only of the Event Control element. The decryption seed (bits k<sub>7</sub>-k<sub>0</sub>) is a one-byte value used to seed the decryption system discussed in ["FM Tag Decryption"](#) (page 471). The 16-bit Region ID value transmitted in block C of the AID group is also transmitted as block D of the Event Control element.

**Note:** The decryption seed must be read from every Event Control message because it may change during the broadcast.

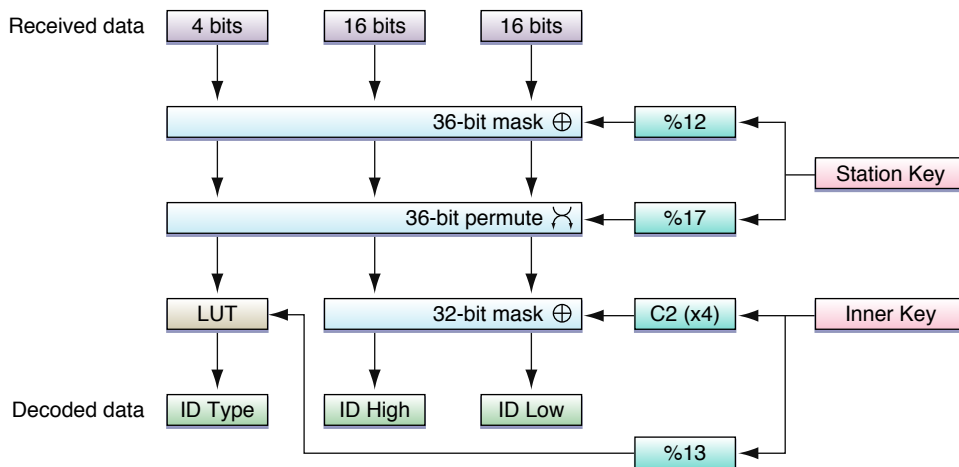
In normal mode, this retransmission lets a receiver quickly verify that Region ID has been received error-free and that element data is being decoded correctly. Retransmission is particularly valuable because the Region ID (as a component of the Station ID) is used as part of the decryption process. In test mode, the Region ID is transmitted only as part of the Event Control element. Retransmission is of less value in test mode because encryption/decryption is disabled.

## FM Tag Decryption

During the transmission of FM tag elements, 36 bits of raw data are passed into an encryption process, including the 4-bit element type ID and 32 bits of element data (ID High and ID Low). The encryption algorithm is keyed by two values generated from RDS data: the Station Key and the Inner Key. The Station Key is the algebraic sum of all three bytes of the Station ID:  $\text{StationKey} = YZ + UV + WX$ . The Inner Key is the linear combination (XOR) of three byte values: the 8 least significant bits of the Station Key, a byte formed from Sequence Length (upper nibble) and Event Counter (lower nibble), and the one-byte Encryption Seed carried by the Event Control element:  $\text{InnerKey} = (\text{StationKey} \& 0xFF) \wedge ((\text{SequenceLength} \ll 4) | \text{EventCounter}) \wedge \text{EncryptionSeed}$ .

The decryption process for FM iTunes tagging elements is diagrammed in [Figure D-6](#) (page 471) and is described below.

**Figure D-6** Element decryption process



To decode the received data, it is first necessary to generate the station key, which is the algebraic sum of all three bytes of the Station ID. ( $\text{StationKey} = YZ + UV + WX$ ). The three input bytes and the result are treated as unsigned integers. The Station ID bytes can be recovered from the AID group.

Upon reception of an ODA group, the 36 bits of coded tag data are passed into the decryption process. The first stage of decoding applies 1 of 12 predefined outer masks to the full 36 bits of received data using a linear mixing (XOR) operation. The 12 predefined masks are listed in [Table D-3](#) (page 472). The mask used is selected as the modulus (integer remainder) of the Station Key divided by 12 ( $\text{MaskRow} = \text{StationKey} \% 12$ ).

**Table D-3** Outer mask lookup table

Row	Bits 35-32	Bits 31-16	Bits 15-0
0	D	D5D0	E3E1
1	2	6073	04B4
2	C	5C59	80D5
3	E	9C6E	78D0
4	3	9E48	2754
5	4	062E	9DB7
6	B	2924	4436
7	6	F308	3628
8	9	9DE8	27A4
9	5	1C71	CEA0
10	8	B5C7	C134
11	7	679C	30BD

The second stage of decoding applies 1 of 17 predefined permute operations to the full 36 bits of received data. The permutation operation reorders the data at the bit level. The 17 permutation operations are shown in [Table D-4](#) (page 472) and [Table D-5](#) (page 473). The operation used is selected as the modulus of the station key divided by 17 ( $\text{PermuteRow} = \text{StationKey} \% 17$ ).

**Table D-4** Permutation lookup table, upper bits

Bits	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18
0	7	11	4	35	3	5	26	2	21	32	6	20	17	0	14	27	9	15
1	13	29	28	12	19	33	32	15	24	27	2	9	4	34	17	8	16	22
2	25	9	24	13	18	11	5	32	22	17	35	16	2	29	6	15	1	19
3	16	20	2	0	28	14	6	25	12	8	7	1	32	24	4	19	33	9
4	10	8	20	7	16	28	33	21	35	1	15	12	25	17	11	22	18	23
5	15	17	16	19	7	3	30	8	28	12	29	34	11	18	25	13	2	31
6	21	31	11	6	34	13	0	33	17	9	26	10	14	7	2	18	22	20
7	4	22	9	15	14	12	1	6	7	21	33	19	35	11	16	29	10	5
8	0	4	35	18	6	7	24	23	11	34	32	3	1	31	22	16	8	30

Bits	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18
9	18	2	15	20	5	22	3	19	4	25	31	26	0	28	13	7	29	1
10	3	0	5	9	17	1	8	18	13	31	4	15	6	12	20	14	23	25
11	9	6	1	33	30	32	21	29	19	13	0	22	10	5	7	2	14	12
12	28	24	26	34	33	18	14	5	25	15	10	0	9	32	19	31	17	13
13	31	5	13	14	0	35	17	12	30	29	22	18	24	25	1	21	4	26
14	22	32	23	24	15	16	35	11	6	33	18	14	28	21	29	0	27	34
15	2	30	0	4	13	15	18	9	8	20	28	32	21	10	31	6	11	35
16	19	33	10	2	27	17	22	7	16	3	21	5	30	20	24	35	32	14

Table D-5 Permutation lookup table, lower bits

Bits	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	8	23	29	30	22	19	16	25	24	10	1	33	12	28	34	18	13	31
1	18	10	31	11	26	3	35	6	25	23	5	0	1	30	7	20	14	21
2	10	8	30	27	23	7	31	12	33	3	0	34	20	21	26	14	28	4
3	5	3	34	31	35	30	21	26	22	15	13	29	18	10	11	23	17	27
4	13	14	2	24	30	29	34	19	26	31	32	4	3	27	9	0	5	6
5	20	6	24	26	10	1	0	23	27	21	4	22	14	5	35	33	32	9
6	1	24	28	4	32	5	15	8	23	29	12	25	30	3	27	19	16	35
7	32	27	20	3	34	31	18	13	30	0	23	28	24	2	25	17	26	8
8	21	2	26	29	12	9	20	17	13	33	25	5	10	14	15	28	27	19
9	27	11	16	8	6	17	24	32	35	9	34	30	21	23	14	12	10	33
10	33	19	27	7	2	21	30	28	34	24	10	16	26	22	29	35	11	32
11	34	31	11	23	20	25	8	27	3	4	15	17	28	26	18	16	35	24
12	35	4	21	6	7	23	29	20	8	30	2	3	27	1	16	11	22	12
13	2	33	7	19	16	20	28	3	32	6	8	27	23	34	10	15	9	11
14	3	26	19	1	31	2	4	5	20	17	30	13	9	12	8	10	7	25
15	26	17	5	12	29	16	27	24	19	34	22	23	25	33	1	7	3	14
16	23	1	8	25	4	26	13	15	31	18	29	12	6	11	0	9	34	28

**Note:** At this stage the outer layer of the encryption system has been reversed and it is now possible to read data from the control elements (upper 4 bits set to 0000 or 1111). The reception and parsing of an event control element (type 0000) is required before the remaining ID oriented elements can be decoded.

The third stage decodes the ID type by passing the upper 4 data bits through a predefined lookup table. It then decodes the ID high and ID low bits by applying an algorithmically-generated inner mask to the lower 32 data bits. The ID lookup is shown in [Table D-6](#) (page 474). The lookup used is selected as the modulus of the inner key divided by 13 ( $IDlookupRow = InnerKey \% 13$ ). The upper 4 data bits select the table column and the decoded ID Type is the cell value at the selected row and column.

**Table D-6** Element type lookup table

ID	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	6	7	2	9	8	11	4	3	10	13	14	1	12	5	15
1	0	3	10	4	5	6	7	11	2	12	9	1	13	14	8	15
2	0	9	3	11	6	7	4	1	14	13	12	5	8	2	10	15
3	0	14	1	9	13	4	12	6	10	11	3	8	2	5	7	15
4	0	7	14	8	3	10	2	12	13	1	5	4	11	9	6	15
5	0	13	11	12	1	14	5	8	4	3	6	10	9	7	2	15
6	0	11	4	13	14	2	9	3	7	8	1	6	5	10	12	15
7	0	2	9	7	10	12	14	13	5	4	8	3	6	1	11	15
8	0	8	13	6	2	9	10	5	11	14	4	12	7	3	1	15
9	0	4	12	1	11	13	8	10	9	5	2	7	14	6	3	15
10	0	5	8	14	7	1	3	9	12	6	11	2	10	4	13	15
11	0	10	6	5	12	11	13	2	1	7	14	9	3	8	4	15
12	0	12	5	10	8	3	1	14	6	2	7	13	4	11	9	15

**Note:** At this point the inner layer of the encryption process for the element type has been reversed and all element types are fully decoded. Decryption of the data bits is accomplished by passing them through the polynomial shown in [Listing D-1](#) (page 475).

To decode ID-oriented elements, it is necessary to generate the inner key. The inner key is a linear mixing (XOR) of three byte values: the 8 least significant bits of the station key, a byte formed from the sequence llength (upper nibble) and event counter (lower nibble) and the encryption seed value, all carried by the event control element ( $InnerKey = (StationKey \& 0xFF) \wedge ((SequenceLength \ll 4) | EventCounter) \wedge EncryptionSeed$ ).

The inner mask is a pseudo-random bit pattern output from a linear feedback shift register seeded by the Inner Key. The Baicheva C2 algorithm is used as the generator polynomial. Four calls are made to the random number generator, each of which produce 8 bits of data. The seed value of the first call is the inner key. The result of the first call becomes the upper 8 bits of the inner mask. The result of the first call also becomes the seed value of the second call, and so on. The result of the fourth call becomes the lower 8 bits of the inner mask.

A code listing for the pseudo-random number generator is shown in [Listing D-1](#) (page 475). To build the 32-bit inner mask, four calls are made to the `IDI_C2` function. The initial call passes in the seed value as the parameter `M`. Passing the output of a call (the return value) in as the seed value of a subsequent call produces the required pseudo-random sequence. Each call produces 8 of the 32 inner mask bits.

**Listing D-1** Pseudo-random number generator

```
int IDI_C2( int M )
{
    int S = 0x0000;

    for ( int i = 0x0080; i > 0; i >>= 1 )
    {
        if ( ( i & M ) != 0 )
        {
            S = ((( S << 1 ) ^ 0x0100 ) & 0x01FF );
        }
        else
        {
            S = (( S << 1 ) & 0x01FF );
        }

        if ( ( S & 0x0100 ) != 0 )
        {
            S = (( S ^ 0x012F ) & 0x01FF );
        }
    }

    return S;
}
```

A reference implementation of the decryption process, in generic source code, is available from Apple upon request.

## Handling Unknown Metadata Types

---

Accessories must support features announced after their product releases by supporting the `UnknownData` plist field. This field is used to pass to iTunes unrecognized data types in the broadcast metadata; iTunes parses the data in the `UnknownData` field and handles it appropriately. In HD broadcasts, unknown metadata may occur in the PSD's Unique File Identifier Data (UFID) fields; in FM broadcasts, it may be passed by the iTunes ODA. On encountering unrecognized ID types in a broadcast, iTunes Tagging accessories must encode the entire unknown data field (the complete UFID in the case of HD and the whole iTunes ODA in the case of FM) in base64 format (64 bytes) and write it to the iPod as `UnknownData`.

The current HD radio UFID data ID types are listed in [Table D-7](#) (page 476). The current FM tag element types are listed in [Table D-2](#) (page 467).

**Table D-7** HD UFID data ID types

ID type	Description
0x3031 or "01"	iTunesSongID
0x3032–0x3033 or "02"–"03"	Reserved
0x3034 or "04"	iTunesAffiliateID
0x3035 or "05"	iTunesStorefrontID
0x3036 or "06"	The contents of the <code>stationURL</code> field is a <code>PodcastFeedURL</code> , not a station URL.

**Note:** The data in the HD radio UFID Identifier field is encoded in conformance with specification ISO-8859-1 and is represented as strings. Refer to iBiquity Document 2174 for more information about parsing the HD UFID data field.

## Resolving Tag Ambiguity

A broadcast tag is deemed ambiguous if the tag button is pressed within 10 seconds before or after a change in the program metadata. In an HD broadcast, a program metadata change is indicated by a Program Service Data (PSD) change; in an FM broadcast, it is indicated by the Event A/B flag. This ambiguity period exists because program metadata changes do not occur exactly on song boundaries, but within 10 seconds before or after that boundary. If the button is pressed less than 10 seconds before the program metadata changes and the new program metadata data is not yet valid, the receiver should generate a nonambiguous tag for the previous data. If a nonambiguous tag is followed by an ambiguous tag of the next song, the result should be two nonambiguous tags. The accessory's UI should not tell the user that a tag is ambiguous.

The accessory must flag ambiguous tags using the `ambiguousTag` plist field. The iTunes application will present a user interface dialog to resolve the ambiguity once the tags have been imported. Accessories should not try to resolve tag ambiguity through their own user interface. Accessories must identify each twin of the ambiguous pair and save this information as a part of the tag data in RAM. Accessories must also identify which twin was active at the time of the tag button press by setting the `ButtonPressed` plist field.

To support tag ambiguity, accessories must store both the current and previous sets of tag data in RAM and update both as program metadata changes occur. A timestamp or timer can be used to mark when either the program metadata changes or the tag button is pressed. The previous and current tags should be stored (when no iPod is connected) or written to the iPod with the `ambiguousTag` plist field set to 1 if these two events occur within 10 seconds of each other.

If the tag button is pressed within 10 seconds **before** a program change:

- The accessory timestamps the button press, or starts a 10 second timer when the tag button is pressed, and sets the `ButtonPressed` byte in the current tag data.
- If a program change occurs within the 10 seconds, the tag data in the current slot is moved to the previous slot and the broadcast tag data fills the current slot. The accessory sets the `ambiguousTag` fields for both tags to 1 and writes both tags to the iPod in a single file.



If the tag button is pressed within 10 seconds **after** a program change:

- The accessory timestamps the program change event, or starts a 10 second timer when the program change occurs, and updates the tag data in the current and previous slots.
- If the tag button is pressed within 10 seconds of the program change, the accessory sets the `ButtonPressed` byte in the current tag data, sets the `ambiguousTag` fields for both tags to 1, and writes both tags to the iPod in a single file.

**Note:** Accessories must resolve tag ambiguity (if it exists) before saving tags or writing them to the iPod. This adds a 10 second delay to the file writing process.

## Capturing and Storing Tag Data

The process of capturing and storing tag data from an HD or FM broadcast occurs in two steps:

1. When the user presses the tag button while listening to the radio accessory, the accessory stores tag data internally.
2. When the user attaches an iPod to the radio accessory the stored tag data is written to the iPod's memory.

**Note:** Tag data must be written to the iPod whenever the iPod is attached, regardless of what mode the accessory may be in. The user must never have to tell the accessory to write tag data nor have to change the accessory's mode to permit tag data to be written.

## Tag Data Writing Process

Each tagged track file written to an iPod consists of an extensible XML dictionary of key-value pairs, formatted as a Mac OS X Core Foundation property list (**plist**). This format is widely used in Mac OS X and can be easily generated and parsed on other platforms. The format is documented at [developer.apple.com/documentation/CoreFoundation/Conceptual/CFPropertyLists/index.html](http://developer.apple.com/documentation/CoreFoundation/Conceptual/CFPropertyLists/index.html). The plist document type definition (DTD) is available at [www.apple.com/DTDs/PropertyList-1.0.dtd](http://www.apple.com/DTDs/PropertyList-1.0.dtd). Listings of typical of plist files can be found in "Sample Tag Files" (page 488).

After authenticating itself to the attached iPod and using `GetiPodOptionsForLingo` to determine that the iPod supports the iTunes Tagging option, an accessory typically uses the following command sequence to write a plist file to the iPod. For details of these commands, see "Lingo 0x0C: Storage Lingo" (page 300). For a sample sequence of actual commands, see Table D-11 (page 484).

Radio accessory	Attached iPod
<code>GetiPodCaps</code>	
	<code>RetiPodCaps</code>
<code>GetiPodFreeSpace</code>	

Radio accessory	Attached iPod
	RetiPodFreeSpace
OpeniPodFeatureFile	
	RetiPodFileHandle
WriteiPodFileData (as many times as needed)	
	iPodACK for each WriteiPodFileData command
CloseiPodFile	
	iPodACK of CloseiPodFileData

The accessory starts by sending a `GetiPodCaps` command to retrieve the iPod's storage lingo capabilities. The iPod responds with the `RetiPodCaps` command, which contains the following data:

- `totalSpace`: the amount of free storage on the iPod.
- `maxFileSize`: the largest possible size of a file on the iPod.
- `maxWriteSize`: the largest amount of data that can be written to the iPod in a single `WriteiPodFileData` command.
- `majorVersion` and `minorVersion`: the version number of the Storage lingo protocol (currently 1.1).

Accessories are required to honor the iPod's `maxWriteSize` limitation; they must never send a `WriteiPodFileData` command with a data payload larger than the iPod's `maxWriteSize` value.

The accessory should also check the iPod's free space before creating a file, to ensure that there is enough free space to write the file. A tag takes approximately 1 KB of data space, so the accessory should verify that the iPod has 1 KB of free space for each tag it intends to write. See ["Note"](#) (page 302). The accessory should also warn the user through its user interface when the iPod does not have enough free space; see ["Accessory User Interface"](#) (page 482) for details.

To create a file on the iPod, the accessory sends an `OpeniPodFeatureFile` command. This command returns a file handle that is used to write data and to close the file. The accessory must send a `WriteiPodFileData` command to write data to the open file. The size of the `WriteiPodFileData` payload is determined by the iPod's `maxWriteSize`; it may take several `WriteiPodFileData` commands to write an entire file. The iPod responds to each `WriteiPodFileData` command with an `iPodACK` command containing the status of the last write. Accessories must verify that each individual write succeeded before sending the next `WriteiPodFileData` command.

Only one tagging file may be opened at a time. Each time a tagging file is opened, a new file is created in `/iPod_Control/Device/Accessories/Tags/` on the iPod (this location may change in future releases). You can look at the files in this directory to confirm that the data you wrote using `WriteiPodFileData` was transferred correctly.

The accessory should close the file using the `CloseiPodFile` command as soon as all the plist data has been written to the iPod. The accessory should verify that the iPod closed the file properly by checking the status of the `iPodACK` command sent in response to the `CloseiPodFile` command. Accessories should never delete tag data stored locally until the `CloseiPodFile` acknowledgment status has been verified.

## Data Transfer to the iPod

Table D-8 (page 479) shows the plist fields an accessory writes to an iPod. The writing process must follow these rules:

- All fields of integer type must be expressed in decimal numbers. Hexadecimal and other non-decimal numbers may not be used.
- Strings in the `Name`, `Artist`, `Album`, `Genre`, and `StationURL` fields must not be truncated; if 64 bytes are received, all 64 must be written. The `StationURL` field may contain 128 bytes.

**Table D-8** Plist fields written to the iPod

Name	Type	Description	HD Source	FM Source
<code>MajorVersion</code>	integer	The major revision level of the file format. This number increments by 1 if there are changes or additions that break compatibility with all prior versions. The current major revision level is 1.	Accessory	Accessory
<code>MinorVersion</code>	integer	The minor revision level of the file format. This number increments by 1 if there are changes or additions that preserve compatibility with prior versions that have the same major revision level. The current minor revision level is 1.	Accessory	Accessory
<code>ManufacturerID</code>	positive integer (decimal format)	Manufacturer IDs are assigned by Apple's Made For iPod program (see note, below).	Accessory	Accessory
<code>ManufacturerName</code>	string	The manufacturer-assigned user-visible name for the manufacturer.	Accessory	Accessory
<code>DeviceName</code>	string	The manufacturer-assigned user-visible name for the device.	Accessory	Accessory
<code>MarkedTracks</code>	array	An array of one or more dictionaries, specifying the data for each tagged track.	Accessory	Accessory
<code>AmbiguousTag</code>	integer	1 to mark an ambiguous track; 0 otherwise.	Accessory	Accessory
<code>ButtonPressed</code>	integer	1 to designate which twin in an ambiguous pair was active when the tag button was pressed; 0 otherwise.	Accessory	Accessory

Name	Type	Description	HD Source	FM Source
Name	string	The name of the track.	HD PSD	RDS: RT+
Artist	string	The name of the artist.	HD PSD	RDS: RT+
Album	string	The name of the album.	HD PSD	RDS: RT+
Genre	string	HD: The program type. The receiver must write this field as received, without decoding. It can contain ID3 codes, 2-digit genre strings, or both.  RDS: If present, the genre data can be extracted from the RT+ ITEM.GENRE class.	HD PSD	RDS: RT+
iTunesSongID	integer	The iTunes song identifier.	HD PSD	RDS: iTunes ODA
iTunesStorefrontID	integer	The iTunes storefront identifier.	HD PSD	RDS: iTunes ODA
StationFrequency	string	The station's frequency, expressed only by digits and an optional decimal point.	HD HOST	Accessory
StationCallLetters	string	HD: The call letters for the station, written exactly as received. Call letters may be up to 12 characters long if the Universal Short Station Name is broadcast.  RDS: Station call letters can be extracted from the RDS PI field or through the RT+ STATIONNAME.SHORT and STATIONNAME.LONG classes.	HD SIS	RDS: PI or RT+
TimeStamp	date	HD: The date and time at which the tagged track was broadcast, in ASCII timestamp format (e.g., 2008-04-16T00:35:42Z). The time zone must be UTC; iTunes converts it into local time. The TimeStamp plist field should be written only if the receiver has received valid, GPS-locked time information over the air and has created a valid timestamp based on it. Receivers using TI DRI352 baseband processor firmware prior to version 17 or NXP SAF355X baseband processor firmware prior to Version 2.1.2.1 should never write this field, because only newer baseband processor versions process the timestamp information correctly. Contact iBiquity for further details.  RDS: Group 4A messages	HD SIS	RDS: Group 4A

Name	Type	Description	HD Source	FM Source
PodcastFeedURL	integer	HD: A value of 1 if the <code>StationURL</code> points to a podcast, 0 otherwise.  RDS: If <code>INFO.URL</code> is filled, the broadcast should be marked as having an associated podcast.	HD PSD	RDS: RT+
StationURL	string	HD: The URL of the station, or of a podcast if <code>PodcastFeedURL</code> is 1.  RDS: <code>PROGRAMME.HOMEPAGE</code>	HD PSD	RDS: RT+
ProgramNumber	integer	HD: The station's multicast channel in the range 1 to 8, where 1 indicates the main program.  RDS: <code>PROGRAMME.SUBCHANNEL</code>	HD HOST	RDS: RT+
iTunesAffiliateID	string	The iTunes affiliate ID, as broadcast.	HD PSD	N/A
iTunesStationID	string	The iTunes station ID, as broadcast.	N/A	RDS: iTunes ODA
UnknownData	data	HD: A field used to pass unknown UFID data to iTunes.  RDS: A field used to pass unknown iTunes Tagging ODA data types to iTunes.  See " <a href="#">Handling Unknown Metadata Types</a> " (page 475)).	HD PSD (UFID data)	RDS: iTunes ODA

**Note:** To obtain a `ManufacturerID`, a partner must submit a product plan to Apple's Made For iPod program. Once the product plan is approved, Apple will respond with a `ManufacturerID` assigned to that plan. Every separate product (i.e., every SKU) must have a unique `ManufacturerID`. The plist file must contain the `ManufacturerID` as a positive integer in decimal format for iTunes to interpret it correctly.

The radio accessory normally writes a tag to the iPod each time the user presses the tag button. However, the tag should not be written if tag ambiguity exists and has not been resolved (see "[Resolving Tag Ambiguity](#)" (page 476)). The opening, writing, and closing of the file should happen in one continuous action to insure that no tag information is lost. Accessories should not keep files open any longer than absolutely necessary. Do not append tags to open files; write a new file to the iPod each time the user presses the tag button. The accessory design must allow tagging while data is being written to the iPod.

The accessory must create a file for each tag saved when the iPod is connected and the tag is not ambiguous. The accessory must write ambiguous tags to the same file when the iPod is connected and the tags are ambiguous. The accessory must write all tags stored locally to one file the next time an iPod is connected, including all ambiguous tags.

**Note:** The `Artist` and `Name` strings are required to store a tag; the `iTunesSongID` integer is desirable but not required.

An accessory must never write a tag more than once. If the user presses the tag button multiple times during the same song, the accessory must write the tag to the iPod after the first button press and then filter out redundant tags by setting a flag to indicate that the broadcast tag data has been written to the iPod.

The accessory must store tags internally when no iPod is connected. If the accessory runs out of storage space, it must prompt the user to connect an iPod. When tags are stored locally, the accessory must write all tag data to a single file the next time an iPod is docked. The `MajorVersion`, `MinorVersion`, `ManufacturerName`, `ManufacturerID`, and `MarkedTracks` fields need to be written only once per file when writing multiple tags to a single file. The rest of the tag data is written in a tag array with each tag designated by the `<dict>` key. "Sample Tag Files" (page 488) shows an example of multiple tags in a single file.

Before it deletes any tag stored locally, the accessory must verify that the iPod has sent an `iPodACK` command in response to each `WriteiPodFileData` command and the `CloseiPodFile` command. The accessory must notify the user that the tags were successfully written to the iPod.

## Accessory User Interface

Table D-9 (page 482) lists the user interface elements that must be used for the tagging feature in a radio accessory. The table shows three sections: Required UI, Optional UI, and UI not allowed. Table D-10 (page 483) lists user interface messages that may appear on the accessory's display.

**Table D-9** Tagging feature user interface implementation

Action		Implementation	
		Speaker	Head unit
Required UI	Indicate tag available	Button (LED) illuminates	Button or logo/type appears
		Logo/type appears (LCD)	Button color or logo/type changes
	Indicate tag captured	Button LED blinks	Button blinks
		Button LED changes color	Button changes color
		Logo/type blinks (LCD)	Graphic
		Message on LCD	Message on screen
		Audible feedback	Audible feedback
	Indicate accessory memory full	Message (LCD)	Message
		Audible feedback	Audible feedback
			Graphic

Action		Implementation	
		Speaker	Head unit
	Indicate iPod memory full	Message (LCD)	Message
		Audible feedback	Audible feedback
Optional UI	Indicate write to iPod	Button LED blinks twice	Button blinks twice
		Message on LCD	Message on screen
		Audible feedback	Audible feedback
	Indicate tag not captured	LED color changes	Button color changes
		Message on LCD	Message on screen
		Audible feedback	Audible feedback
	Tags remaining	Message on LCD	Message on screen
UI not allowed	Resolving tag ambiguity		
	Choosing whether or not to write tags to the iPod		

**Table D-10** Tagging feature UI text messages

Condition	Message
Capturing tags	"Tag available"
	"Tag stored"
	"Tag stored. XX remaining."
Accessory memory full	"Memory full. Connect iPod."
	"Connect iPod to transfer tags."
iPod memory full	"iPod full. Tags cannot be stored."
Write to iPod	"Tags transferred to iPod."
	"Tags saved to iPod. XX remaining."

**Note:** These message texts are preliminary and are provided for guidance only. The specific texts to be displayed are expected to change as the tagging feature user interface is further defined.

The accessory should not have a UI function that deletes individual tags; this is done by iTunes. A function to delete all tags is acceptable as a way to restore the accessory to its factory settings.

## Tag Button Text

The button on the radio accessory used to tag a song should bear the word “Tag.” The word “Tag” should be presented in a manner consistent with the text on the device’s other buttons (the same font, weight, capitalization, color, illumination, and so on).

## Sample Command Sequence

[Table D-11](#) (page 484) shows a sample sequence of commands that implements radio tagging in an accessory.

**Table D-11** Radio tagging command sequence

Step	Accessory command	iPod command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> <li>■ If the accessory receives no ACK command within 1 second, it may retry Step 1 at 1-second intervals as long as iPod Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.</li> <li>■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the iPod does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83). A sample command sequence is listed in <a href="#">Table F-27</a> (page 545).</li> <li>■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.</li> </ul>			
3	GetiPodOptions-ForLingo		getting options for General Lingo
4		RetiPodOptions-ForLingo	returning options of 000000000003F3FF (Line out usage   Video output   NTSC video signal format   PAL video signal format   Composite video out connection   S-Video video out connection   Component video out connection   Closed captioning (video)   Video aspect ratio 4:3 (fullscreen)   Video aspect ratio 6:9 (widescreen)   reserved   app communication capable   iPod notifications) for General Lingo



Step	Accessory command	iPod command	Comment
5	GetiPodOptions-ForLingo		getting options for Storage Lingo
6		RetiPodOptions-ForLingo	returning options of 0000000000000003 (iTunes tagging   Nike + iPod cardio equipment) for Storage Lingo
7	SetFIDTokenValues		setting 8 FID tokens ; IdentifyToken = (lingoes: 0/12   options 0x00000002   device ID 0x00000200); AccCapsToken = 0x0000000000000805; AccInfoToken = Acc name (Radio); AccInfoToken = Acc FW version (v1.0.1); AccInfoToken = Acc HW version (v1.0.0); AccInfoToken = Acc manufacturer (Radio Manufacturer); AccInfoToken = Acc model number (M78901LL/Z); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'selected')
8		RetFIDTokenValueACKs	8 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; iPodPreferenceToken = (line out usage) accepted
9	EndIDPS		status 'finished with IDPS; proceed to authentication'
10		IDPSStatus	status 'ready for auth'
11		GetDevAuthentication-Info	no params
12	RetDevAuthentication-Info		returning auth protocol v2.0; current section index: 0; maximum section index: 1; cert data ...
13		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
14	RetDevAuthentication-Info		returning auth protocol v2.0; current section index: 1; maximum section index: 1; cert data ...

Step	Accessory command	iPod command	Comment
15		AckDevAuthentication-Info	acknowledging 'auth info supported'
16		GetDevAuthentication-Signature	offering challenge ...
17	RetDevAuthentication-Signature		returning signature ...
18		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
19	GetiPodCaps		no params
20		RetiPodCaps	returning 'total space 2147483648; max file size 1073741824; max write size 500; v1.02'
21	GetiPodFreeSpace		no params
22		RetiPodFreeSpace	returning 130023424 bytes
23	OpeniPodFeatureFile		opening feature type 'Radio Tagging'
24		RetiPodFileHandle	returning file handle 0
25	WriteiPodFileData		writing 184 bytes at offset 0 and handle 0: <?xml version=""1.0" encoding=""UTF-8""?> <!DOCTYPE plist PUBLIC ""-//Apple Inc//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd""> <plist version=""1.0""> <dict>
26		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
27	WriteiPodFileData		writing 290 bytes at offset 185 and handle 0: <key>DeviceName</key> <string>IES2</string> <key>MajorVersion</key> <integer>1</integer> <key>MinorVersion</key> <integer>0</integer> <key>ManufacturerID</key> <integer>17</integer> <key>ManufacturerName</key> <string>Polk Audio</string> <key>MarkedTracks</key> <array> <dict>

Step	Accessory command	iPod command	Comment
28		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
29	WriteiPodFileData		writing 472 bytes at offset 476 and handle 0: <key>Name</key> <string>The Rising</string> <key>Artist</key> <string>Bruce Springsteen</string> <key>Album</key> <string>The Rising</string> <key>StationURL</key> <string>http://www.hdradio.com</string> <key>iTunesSongID</key> <integer>192903160</integer> <key>StationCallLetters</key> <string>IHDR</string> <key>StationFrequency</key> <string>88.1 FM</string> <key>StreamID</key> <string>2</string> <key>ProgramType</key> <string>Classic Rock</string>
30		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
31	WriteiPodFileData		writing 210 bytes at offset 949 and handle 0: <key>TimeStam</key> <date>2009-06-01T09:45:57Z</date> <key>iTunesStorefrontID</key> <integer>143441</integer> <key>iTunesPlaylistID</key> <integer>192901525</integer> </dict> </array> </dict> </plist>
32		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
33	CloseiPodFile		closing file with handle 0
34		iPodAck	acknowledging 'Success' to command 'Storage Lingo::CloseiPodFile' with file handle 0

## Sample Tag Files

This section lists sample plist files that might be generated by both HD radio tagging and FM radio tagging.

### HD Radio Samples

---

The following is a sample HD radio plist-formatted tag file showing a header and one stored tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>MajorVersion</key>
  <integer>1</integer>
  <key>MinorVersion</key>
  <integer>1</integer>
  <key>ManufacturerID</key>
  <integer>17</integer>
  <key>ManufacturerName</key>
  <string>Acme</string>
  <key>DeviceName</key>
  <string>AC-HDR100</string>
  <key>MarkedTracks</key>
  <array>
    <dict>
      <key>Name</key>
      <string>Times Like These</string>
      <key>Artist</key>
      <string>Foo Fighters</string>
      <key>Album</key>
      <string>One by One</string>
      <key>iTunesSongID</key>
      <integer>6906304</integer>
      <key>iTunesStorefrontID</key>
      <integer>143441</integer>
      <key>StationFrequency</key>
      <string>104.9</string>
      <key>StationCallLetters</key>
      <string>KCNL</string>
      <key>PodcastFeedURL</key>
      <string>0</string>
      <key>StationURL</key>
      <string>http://www.channel1049.com</string>
      <key>Genre</key>
      <string>(20) Alternative</string>
      <key>TimeStamp</key>
      <date>2007-04-16T00:35:42Z</date>
      <key>ProgramNumber</key>
      <integer>1</integer>
      <key>iTunesAffiliateID</key>
      <string>00-12uR34oIcpQ</string>
      <key>UnknownData</key>
      <data>MDcwNDEyMzQwMTAyMTEwNDA4MTJhYjU2Z2g=</data>
```

## iTunes Tagging

```

    </dict>
  </array>
</dict>
</plist>

```

The following is a sample HD radio plist-formatted tag file showing how ambiguous tags are saved to an iPod:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>MajorVersion</key>
  <integer>1</integer>
  <key>MinorVersion</key>
  <integer>1</integer>
  <key>ManufacturerID</key>
  <integer>17</integer>
  <key>ManufacturerName</key>
  <string>Acme</string>
  <key>DeviceName</key>
  <string>AC-HDR100</string>
  <key>MarkedTracks</key>
  <array>
    <dict>
      <key>AmbiguousTag</key>
      <integer>1</integer>
      <key>ButtonPressed</key>
      <integer>1</integer>
      <key>Name</key>
      <string>Times Like These</string>
      <key>Artist</key>
      <string>Foo Fighters</string>
      <key>Album</key>
      <string>One by One</string>
      <key>iTunesSongID</key>
      <integer>6906304</integer>
      <key>iTunesStorefrontID</key>
      <integer>143441</integer>
      <key>StationFrequency</key>
      <string>104.9</string>
      <key>StationCallLetters</key>
      <string>KCNL</string>
      <key>PodcastFeedURL</key>
      <string>0</string>
      <key>StationURL</key>
      <string>http://www.channel1049.com</string>
      <key>Genre</key>
      <string>(20) Alternative</string>
      <key>TimeStamp</key>
      <date>2007-04-16T00:35:42Z</date>
      <key>ProgramNumber</key>
      <integer>1</integer>
      <key>iTunesAffiliateID</key>
      <string>00-12uR34oIcpQ</string>
    </dict>
  </dict>
</plist>

```

## iTunes Tagging

```

        <key>AmbiguousTag</key>
        <integer>1</integer>
        <key>ButtonPressed</key>
        <integer>0</integer>
        <key>Name</key>
        <string>Vertigo</string>
        <key>Artist</key>
        <string>U2</string>
        <key>Album</key>
        <string>How to Dismantle an Atomic Bomb</string>
        <key>iTunesSongID</key>
        <integer>29600235</integer>
        <key>iTunesStorefrontID</key>
        <integer>143441</integer>
        <key>StationFrequency</key>
        <string>104.9</string>
        <key>StationCallLetters</key>
        <string>KCNL</string>
        <key>PodcastFeedURL</key>
        <string>0</string>
        <key>StationURL</key>
        <string>http://www.channel1049.com</string>
        <key>Genre</key>
        <string>(17) Rock</string>
        <key>TimeStamp</key>
        <date>2007-04-16T00:35:45Z</date>
        <key>ProgramNumber</key>
        <integer>1</integer>
        <key>iTunesAffiliateID</key>
        <string>00-12uR34oIcpQ</string>
    </dict>
</array>
</dict>
</plist>

```

The following is a sample HD radio plist-formatted tag file showing how an accessory that has multiple tags stored locally writes those tags to an iPod:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>MajorVersion</key>
    <integer>1</integer>
    <key>MinorVersion</key>
    <integer>1</integer>
    <key>ManufacturerID</key>
    <integer>17</integer>
    <key>ManufacturerName</key>
    <string>Acme</string>
    <key>DeviceName</key>
    <string>AC-HDR100</string>
    <key>MarkedTracks</key>
    <array>
        <dict>
            <key>Name</key>
            <string>Times Like These</string>

```

## iTunes Tagging

```

    <key>Artist</key>
    <string>Foo Fighters</string>
    <key>Album</key>
    <string>One by One</string>
    <key>iTunesSongID</key>
    <integer>6906304</integer>
    <key>iTunesStorefrontID</key>
    <integer>143441</integer>
    <key>StationFrequency</key>
    <string>104.9</string>
    <key>StationCallLetters</key>
    <string>KCNL</string>
    <key>PodcastFeedURL</key>
    <string>0</string>
    <key>StationURL</key>
    <string>http://www.channel1049.com</string>
    <key>Genre</key>
    <string>(20) Alternative</string>
    <key>TimeStamp</key>
    <date>2007-04-16T00:35:42Z</date>
    <key>ProgramNumber</key>
    <integer>1</integer>
    <key>iTunesAffiliateID</key>
    <string>00-12uR34oIcpQ</string>
</dict>
<dict>
    <key>Name</key>
    <string>Vertigo</string>
    <key>Artist</key>
    <string>U2</string>
    <key>Album</key>
    <string>How to Dismantle an Atomic Bomb</string>
    <key>iTunesSongID</key>
    <integer>29600235</integer>
    <key>iTunesStorefrontID</key>
    <integer>143441</integer>
    <key>StationFrequency</key>
    <string>104.9</string>
    <key>StationCallLetters</key>
    <string>KCNL</string>
    <key>PodcastFeedURL</key>
    <string>0</string>
    <key>StationURL</key>
    <string>http://www.channel1049.com</string>
    <key>Genre</key>
    <string>(17) Rock</string>
    <key>TimeStamp</key>
    <date>2007-04-16T00:43:45Z</date>
    <integer>1</integer>
    <key>ProgramNumber</key>
    <integer>1</integer>
    <key>iTunesAffiliateID</key>
    <string>00-12uR34oIcpQ</string>
</dict>
<dict>
    <key>Name</key>
    <string>Ball and Chain</string>
    <key>Artist</key>

```

```

        <string>Social Distortion</string>
        <key>Album</key>
        <string>Social Distortion</string>
        <key>iTunesSongID</key>
        <integer>197986000</integer>
        <key>iTunesStorefrontID</key>
        <integer>143441</integer>
        <key>StationFrequency</key>
        <string>104.9</string>
        <key>StationCallLetters</key>
        <string>KCNL</string>
        <key>PodcastFeedURL</key>
        <string>0</string>
        <key>StationURL</key>
        <string>http://www.channel1049.com</string>
        <key>Genre</key>
        <string>(17) Rock</string>
        <key>TimeStamp</key>
        <date>2007-04-16T00:48:45Z</date>
        <key>ProgramNumber</key>
        <integer>1</integer>
        <key>iTunesAffiliateID</key>
        <string>00-12uR34oIcpQ</string>
    </dict>
</array>
</dict>
</plist>

```

## FM Radio Sample

---

The following is a sample FM radio plist-formatted tag file showing a header and one stored tag:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>MajorVersion</key>
    <integer>1</integer>
    <key>MinorVersion</key>
    <integer>1</integer>
    <key>ManufacturerID</key>
    <integer>17</integer>
    <key>ManufacturerName</key>
    <string>Acme</string>
    <key>DeviceName</key>
    <string>AC-HDR100</string>
    <key>MarkedTracks</key>
    <array>
        <dict>
            <key>Name</key>
            <string>Times Like These</string>
            <key>Artist</key>
            <string>Foo Fighters</string>
            <key>Album</key>
            <string>One by One</string>
            <key>iTunesSongID</key>

```



## iTunes Tagging

```

        <integer>6906304</integer>
        <key>iTunesStorefrontID</key>
        <integer>143441</integer>
        <key>StationFrequency</key>
        <string>104.9</string>
        <key>StationCallLetters</key>
        <string>KCNL</string>
        <key>StationURL</key>
        <string>http://www.channel1049.com</string>
        <key>Genre</key>
        <string>(20) Alternative</string>
        <key>TimeStamp</key>
        <date>2007-04-16T00:35:42Z</date>
        <key>ProgramNumber</key>
        <integer>1</integer>
        <key>iTunesStationID</key>
        <string>10491557</string>
        <key>UnknownData</key>
        <data>MDcwNDEyMzQwMTAyMTEwNDA4MTJhYjU2Z2g=</data>
    </dict>
</array>
</dict>
</plist>

```



# Nike + iPod Cardio Equipment System

---

The Nike + iPod cardio equipment feature lets an iPod user record workout data gathered by an attached sports or exercise device, while at the same time enjoying entertainment from the iPod.

An iPod can support only one such device at any time, and that device must be attached using the 30-pin connector. Currently the 2nd generation iPod touch and the 3rd and 4th generation iPod nanos support this system.

## Using Nike + iPod Cardio Equipment

To use the capabilities of the Nike + iPod cardio equipment feature, an iPod user typically does the following:

1. The user connects a compatible iPod to a compatible piece of cardio equipment, makes an entertainment selection, and selects either QuickStart or a specific workout type.
2. The cardio equipment downloads user preferences and other information from the iPod.
3. If there is enough space on the iPod to store the workout data, the cardio equipment displays the message “Recording workout to iPod.” It opens a file and records data in real time.
4. The cardio equipment prompts the user to accept the weight setting retrieved from the iPod or change it, in which case it stores the new weight on the iPod.
5. When the workout ends, the cardio equipment closes the file on the iPod and displays a closing message with a workout summary. It confirms that the workout has been recorded and tells the user how to view the results.
6. The user connects the iPod to a computer and iTunes uploads the workout files to [nikeplus.com](http://nikeplus.com), where the user can view the results.

## Cardio Equipment Design

This section specifies the requirements for cardio equipment that works with the Nike + iPod system.

### Determining iPod Support for Cardio Equipment

---

To use the Nike + iPod feature, the cardio equipment must first determine that the connected iPod supports the required minimum lingo versions shown in [Table E-1](#) (page 496).

**Table E-1** Lingo version support

Lingo	Minimum Required Protocol Version
0x09: Sports	v1.01
0x0C: Storage	v1.02

If the minimum required protocol versions are supported by the connected iPod, the cardio equipment must next identify itself as supporting those lingoes; see [“Identification Procedure”](#) (page 496).

The final step is for the cardio equipment to query the Sports lingo capabilities of the iPod with a `GetiPodOptionsForLingo` command and then parse the resulting `RetiPodOptionsForLingo` command for the Nike + iPod Cardio Equipment bit (0x01); see [“Command 0x4C: RetiPodOptionsForLingo”](#) (page 140).

**Note:** All Sports lingo commands require authentication. The cardio equipment must not send the `GetiPodOptionsForLingo` command until after the authentication process has begun. See [“iPod Authentication of Device”](#) (page 55) for further information.

## Identification Procedure

The cardio equipment must perform an identification procedure by which it discovers the capabilities of the connected iPod. If it attempts to identify for a lingo not supported by the attached iPod, the identification fails and the iPod displays an error message.

The following identification procedure is recommended:

1. Send a `StartIDPS` command. If the iPod responds with an `ACK` command passing 0x04 (Bad Parameter), the accessory must send `IdentifyDeviceLingoes` instead, with `deviceId=0x0000` and identifying only the General lingo with no options.
2. Query the iPod for the version numbers of all necessary lingoes.
3. Based on the responses to the query, form appropriate FID values, declare them for the required lingoes, and request authentication if necessary.
4. End the IDPS process by sending `EndIDPS`.

## Sample Identification Processes

This section presents two sample identification sequences for a cardio equipment accessory. In both cases the sequences begin with the accessory sending a `StartIDPS` command. However, in both cases the iPod does not support IDPS, so the accessory reverts to sending `IdentifyDeviceLingoes`. For examples of identification sequences in which IDPS is supported, see [“Sample Identification Sequences”](#) (page 334).

For a first example, consider a piece of cardio equipment that is intended to offer simple iPod playback control (play/pause, next/previous track, etc.), display information about the currently playing track, and log workout data to the iPod. To achieve these goals, the cardio equipment needs access to four iAP lingoes:

- Simple Remote

- Display Remote
- Sports
- Storage

The example in [Table E-2](#) (page 497) shows the command traffic exchanged between the cardio equipment and an iPod Classic running software v1.1.1.

**Table E-2** Sample identification process 1

cardio equipment	iPod Classic (v1.1.1)
StartIDPS	
	ACK( StartIDPS, 0x04 = Bad Parameter)
IdentifyDeviceLingoes (Lingoes: 0x0001, Options: 0x00, DeviceID: 0x0000)	
	ACK( IdentifyDeviceLingoes, Success)
RequestLingoProtocolVersion(0x02)	
	ReturnLingoProtocolVersion(0x02, v1.02)
RequestLingoProtocolVersion(0x03)	
	ReturnLingoProtocolVersion(0x03, v1.05)
RequestLingoProtocolVersion(0x09)	
	ACK( RequestLingoProtocolVersion, Bad Parameter) (This iPod does not support the Sports lingo.)
RequestLingoProtocolVersion (0x0C)	
	ReturnLingoProtocolVersion(0x0C, v1.01) (This iPod does not support the required version of the Storage lingo.)
IdentifyDeviceLingoes(Lingoes:0x000D, Options:0x02, DeviceID:0x0200)	
	ACK( IdentifyDeviceLingoes, Success)
	GetDeviceAuthenticationInfo()

In this first example, the cardio equipment determines that the attached iPod does not support the proper versions of the Sports and Storage lingoes and thus does not attempt to identify for those lingoes. Display Remote and Simple Remote are both supported, and the cardio equipment properly identifies and authenticates for those lingoes.

A more successful example is shown in [Table E-3](#) (page 498). It lists the command traffic exchanged between the cardio equipment and a 3G iPod nano updated to support the Nike + iPod feature.

**Table E-3** Sample identification process 2

Cardio equipment	3G iPod nano (version 1.1.2)
StartIDPS	
	ACK( StartIDPS, 0x04 = Bad Parameter)
IdentifyDeviceLingoes (Lingoes:0x0001, Options:0x00, DeviceID:0x0000)	
	ACK(IdentifyDeviceLingoes, Success)
RequestLingoProtocolVersion(0x02)	
	ReturnLingoProtocolVersion(0x02, v1.02)
RequestLingoProtocolVersion(0x03)	
	ReturnLingoProtocolVersion(0x03, v1.05)
RequestLingoProtocolVersion(0x09)	
	ReturnLingoProtocolVersion(0x09, v1.01)
RequestLingoProtocolVersion(0x0C)	
	ReturnLingoProtocolVersion(0x0C, v1.02)
IdentifyDeviceLingoes(Lingoes:0x120D, Options:0x02, DeviceID: 0x0200)	
	ACK(IdentifyDeviceLingoes, Success)
	GetDeviceAuthenticationInfo()

In this second example, the iPod supports the proper minimum protocol versions for each of the necessary lingoes. The cardio equipment identifies for all of those lingoes and operation proceeds normally.

## Declaring Cardio Equipment Support to the iPod

In the future, the iPod may change its user interface elements when attached to Nike + iPod cardio equipment. Hence, the cardio equipment must declare its support of the Nike + iPod feature. This is also necessary so that the iPod can differentiate between types of Sports lingo devices (e.g. between a Nike + iPod Sports Kit Receiver and a stair climber).

The cardio equipment declares itself as such by supporting the Sports lingo `GetDeviceCaps` command and replying to it with a `RetDeviceCaps` command with `capsMask` bit 9 = 1, indicating to the iPod that the cardio equipment supports the required commands.

## Accessing User Data

Nike + iPod compatible cardio equipment can access certain user data stored on an iPod. See [“Sports Lingo commands”](#) (page 277) for detailed information on the commands available for retrieving and/or setting the available user data fields.

Proper use of this user data requires that the cardio equipment must:

1. Always get/set data for the current `userIndex`. It is assumed that the iPod is already configured for the correct `userIndex`.
2. Write `userData` fields back to the iPod if, and only if, the user entered new or updated information via the cardio equipment user interface. For example, if the cardio equipment uses a default value for weight during a workout, that weight must not be written back to the `userData` on the iPod.
3. Never display the name field for `userIndex = 0`. That value is reserved for the default new or unknown user; consequently, the associated name field is invalid. The `userData` name fields for other `userIndex` values should be valid.

## Recording Workout Data

Workout data is stored on the iPod as a UTF-8 encoded, XML-formatted file. The cardio equipment is responsible for writing the data directly to the iPod's file system via the Storage lingo. Appropriate XML elements are written to the file in real time as the workout proceeds.

### Writing Workout Data

After successfully providing valid device authentication information to the attached iPod, the cardio equipment typically uses the following command sequence to write the workout data file to the iPod. For details of these commands, see [“Lingo 0x09: Sports Lingo”](#) (page 277) and [“Lingo 0x0C: Storage Lingo”](#) (page 300).

**Table E-4** Writing workout data to the iPod

Cardio equipment	iPod (with Nike + iPod feature support)
<code>SportsLingo: GetiPodCaps()</code>	
	<code>SportsLingo: RetiPodCaps()</code>
<code>StorageLingo: GetiPodCaps()</code>	
	<code>StorageLingo: RetiPodCaps()</code>
<code>OpeniPodFeatureFile( featureType:0x02, optionsMask:0x000B, fileData:"&lt;/gymData&gt;")</code>	
	<code>RetiPodFileHandle()</code>
<code>WriteiPodFileData()</code> as many times as needed	

Cardio equipment	iPod (with Nike + iPod feature support)
	iPodACK(WriteiPodFileData, status) for each WriteiPodFileData command
CloseiPodFile()	
	iPodACK(CloseiPodFile, ackStatus)

The cardio equipment starts by sending a `GetiPodCaps` command to retrieve the Sports lingo capabilities of the iPod. The iPod responds with a `RetiPodCaps` command.

If the iPod indicates that it supports the Nike + iPod feature, the cardio equipment next sends a `GetiPodCaps` command to determine the Storage lingo capabilities of the iPod. The iPod responds with a `RetiPodCaps` command containing the following data:

- `totalSpace`: the amount of storage on the iPod, including space currently in use
- `maxFileSize`: the largest possible size of a file on the iPod
- `maxWriteSize`: the largest amount of data that can be written to the iPod in a single `WriteiPodFileData` command
- `majorVersion` and `minorVersion`: the version number of the Storage lingo protocol (currently 1.02)

The cardio equipment is required to honor the `maxWriteSize` limitation of the iPod; it must never send a `WriteiPodFileData` command with a data payload larger than that `maxWriteSize` value.

To create a cardio workout file on the iPod, the cardio equipment sends an `OpeniPodFeatureFile` command with `feature = 0x02`, `options = 0x000B`, and `fileData` equal to the UTF-8 representation of the string necessary to close the root element: `"</gymData>"` or `0x3C 0x2F 0x67 0x79 0x6D 0x44 0x61 0x72 0x61 0x3E`. On receiving this command, the iPod performs the following steps:

1. Creates a new workout file in `/iPod_Control/Device/Trainer/Workouts/Empeds/USERNAME/latest/` (where `USERNAME` is the name of the current `userIndex`)
2. Sends a `RetiPodFileHandle` command with a handle to the new file

Next, the cardio equipment must use a series of `WriteiPodFileData` commands to write the following elements to the file:

```

the XML version header line
root element opening tag: <gymData>
<vers>
<equipmentInfo>
<userInfo>
<template> (optional)
<interval> with <event> "start" or "continue" and initial values for all child elements

```

See ["XML Element Formats"](#) (page 502) for details about valid XML elements.



As the workout progresses, the cardio equipment is required to write `<interval>` XML elements on a real-time basis at least every 10 seconds. Certain child elements of `<interval>` (the `<incline>` element, for example) are required only if their value has changed since the previous `<interval>` element .

**Note:** All `<interval>` elements written to file must be complete and closed. Because a user can detach the iPod at any time during the process, `<interval>` elements must not be broken across multiple `WriteiPodFileData` commands.

At the end of the workout, the cardio equipment must write the `<workoutSummary>` element to the file and then send the `CloseiPodFile` command. Upon receiving the `CloseiPodFile` command, the iPod performs the following actions (assuming the proper option bits were set in `OpeniPodFeatureFile`):

1. Append the `<iPodInfo>` XML element to file.
2. Write the specified `fileData` to close the root element.
3. Close the XML file.
4. Compute and insert an XML `<Signature>` element in to file.
5. Send an `iPodACK` command acknowledging success.

While performing these steps, the iPod is unresponsive to further iAP commands. Signature calculation requires many seconds for long workouts. The cardio equipment must wait for the `CloseiPodFile` command to fully complete before sending any new iAP commands. The `CloseiPodFile` command is complete when the iPod returns an `iPodACK` with any status other than command pending (0x06).

## Workout Events

---

Events that occur as part of a normal workout must be tracked and recorded to the XML file as part of the `<event>` child element of `<interval>`. There are five event types to be recorded:

- **start:** the first `<interval>` of a new workout
- **continue:** the first `<interval>` of a workout already in progress
- **pause:** when a user pauses the active workout
- **resume:** when a user resumes the workout
- **end:** the final `<interval>` of a completed workout

Workout time must not elapse between a pause event and a resume event; the `<sec>` children of both the pause and the resume `<interval>` elements must be identical. Continue events are used to mark a file that was opened after the workout has already started; some amount of `<interval>` data from the beginning of the workout is not present in the file.

## File System Full

The iPod will determine whether or not enough disk space is available to record and sign the workout data. When the iPod file system no longer contains enough free space, the iPod will respond with `iPodACK = (0x03)` (out of resources) to any `OpeniPodFeatureFile` or `WriteiPodFileData` commands. If this occurs during an `OpeniPodFeatureFile` command, the cardio equipment must notify the user that the iPod is full and that the workout will not be recorded.

If an “out of resources” `iPodACK` is received in response to a `WriteiPodFileData`, the cardio equipment must close the current workout file without writing an end `<interval>` or `<workoutSummary>` and must alert the user that the iPod is full and that the workout is no longer being recorded.

## File Writing Error Recovery

Cardio equipment accessories must properly handle unexpected failures during the file writing process. In the event of a failure while writing data to the iPod (as indicated by lack of an `ackStatus=Success` in response to `WriteiPodFileData`), the cardio equipment should first retry the operation. If the operation fails twice, the current workout file must be closed and the cardio equipment must attempt to open a new workout file.

## Recording a Workout Already in Progress

When opening a new workout data file mid-workout, the standard procedure for creating a new workout file must be followed. The difference between a standard workout file and one started in the middle of a workout is that the first `<interval>` element will have a nonzero `<sec>` value and an `<event>` continue (instead of start) value. It is the responsibility of the end consumer of the data (such as `nikeplus.com`) to recognize a workout file that started mid-workout and act appropriately.

## XML Element Formats

All file data must be well-formed and valid XML. Character entities are not currently supported by the iPod. All special characters (`&`, `<`, `>`) used as element data content must be encapsulated as part of a CDATA section. For human readability, it is recommended that each element close tag be followed by a newline character (`0x0A`). XML elements are required to appear in the order shown in the table with two exceptions:

- The `<userInfo>` element may appear at any point in the file.
- The `<template>` element may appear multiple times and at any point in the file.

All `<interval>` elements must be written in chronological order (ascending `<sec>` values). All distances must be recorded to 0.01 km resolution. All kilocalorie values must be recorded to 0.01 kCal resolution. In situations where the user is not moving and the workout speed is essentially zero, the `<pace>` element should be set to 0 instead of to infinity.

**Table E-5** XML element usage

Element	Format	Example	Tread-mill	Ellip-tical	Stepper	Bike	Other
Header							

Element	Format	Example	Tread-mill	Ellip-tical	Stepper	Bike	Other
XML Header		<?xml version="1.0" encoding="UTF-8"?>	Req	Req	Req	Req	Req
Root Element Open		<gymData>	Req	Req	Req	Req	Req
File Format Version	integer	<vers>1</vers>	Req	Req	Req	Req	Req
<b>Equipment Info</b>							
		<equipmentInfo>	Req	Req	Req	Req	Req
Manufacturer ID	integer (hex format)	<manufacturerID>0123ABCD</manufacturerID> (see Note below)	Req	Req	Req	Req	Req
Manufacturer Name	string	<manufacturerName>Partner A</manufacturerName>	Req	Req	Req	Req	Req
Equipment Type	string enum: Treadmill, Elliptical, Stepper, Bike, or Other	<type>Treadmill</type>	Req	Req	Req	Req	Req
Equipment Model	string	<model> ExerPro 5000</model>	Req	Req	Req	Req	Req
Equipment Serial No.	string	<serialNumber>XYZ012345TG88</serialNumber>	Opt	Opt	Opt	Opt	Opt
Gym Name	string	<gymName>GloboGym</gymName>	Opt	Opt	Opt	Opt	Opt
Gym Location	string	<gymLocation>Cupertino, CA</gymLocation>	Opt	Opt	Opt	Opt	Opt
		</equipmentInfo>	Req	Req	Req	Req	Req
<b>User Info</b>							
		<userInfo>	Req	Req	Req	Req	Req
User Name	string	<name>Jane Doe</name>	Opt	Opt	Opt	Opt	Opt

Element	Format	Example	Tread-mill	Ellip-tical	Stepper	Bike	Other
Weight	float kilograms	<kg>53.5</kg>	Req	Req	Req	Opt	Opt
Gender	string enum: female or male	<gender>female </gender>	Opt	Opt	Opt	Opt	Opt
		</userInfo>	Req	Req	Req	Opt	Opt
<b>Workout Template</b>							
		<template>	Opt	Opt	Opt	Opt	Opt
Template Name	string	<templateName>Hills </templateName>	Opt	Opt	Opt	Opt	Opt
Caloric Goal	float kilocalories	<kCal>200.00</kCal>	Opt	Opt	Opt	Opt	Opt
Time Goal	integer seconds	<sec>900</sec>	Opt	Opt	Opt	Opt	Opt
Distance Goal	float kilometers	<km>10.00</km>	Opt	Opt	N/A	Opt	Opt
	float floors	<floors>30.0</floors>	N/A	N/A	Opt	N/A	Opt
Speed Goal	integer seconds per kilometer	<pace>450</pace>	Opt	N/A	N/A	N/A	Opt
	integer steps/strides per minute	<spm>40</spm>	N/A	Opt	Opt	N/A	Opt
	integer revolutions per minute	<rpm>60</rpm>	N/A	N/A	N/A	Opt	Opt
Heart Rate Goal	integer beats per minute	<bpm>140</bpm>	Opt	Opt	N/A	Opt	Opt
		</template>	Opt	Opt	Opt	Opt	Opt
<b>Interval Data</b> (written at least every 10 seconds)							
		<interval>	Req	Req	Req	Req	Req
Event Type	string enum: start, continue, pause, resume, or end	<event>pause </event>	Opt	Opt	Opt	Opt	Opt
Current Elapsed Time	integer seconds	<sec>1200</sec>	Req	Req	Req	Req	Req
Current Calories	float kilocalories	<kCal>230.12</kCal>	Req	Req	Req	Req	Req

Element	Format	Example	Tread-mill	Ellip-tical	Stepper	Bike	Other
Current Distance	float kilometers	<km>3.25</km>	Req	Req	N/A	Req	Opt
	float floors	<floors>11.0</floors>	N/A	N/A	Req	N/A	Opt
Current Speed	integer seconds per kilometer	<pace>430</pace>	Req	N/A	N/A	N/A	Opt
	integer steps/strides per minute	<spm>40</spm>	N/A	Req	Req	N/A	Opt
	integer revolutions per minute	<rpm>55</rpm>	N/A	N/A	N/A	Req	Opt
Current Heart Rate	integer beats per minute	<bpm>101</bpm>	Opt	Opt	Opt	Opt	Opt
Incline	float percent	<incline>10.1</incline>	Required only if value has changed				
Resistance/Effort	integer level	<level>3</level>	Required only if value has changed				
		</interval>	Req	Req	Req	Req	Req
<b>Workout Summary</b>							
		<workoutSummary>	Req	Req	Req	Req	Req
Total Elapsed Time	integer seconds	<sec>2450</sec>	Req	Req	Req	Req	Req
Total Calories	float kilocalories	<kCal>342.34</kCal>	Req	Req	Req	Req	Req
Total Distance	float kilometers	<km>10.15</km>	Req	Req	N/A	Req	Opt
	float floors	<floors>11.0</floors>	N/A	N/A	Req	N/A	Opt
Average Speed	integer seconds per kilometer	<pace>430</pace>	Req	N/A	N/A	N/A	Opt
	integer steps/strides per minute	<spm>40</spm>	N/A	Req	Req	N/A	Opt
	integer revolutions per minute	<rpm>55</rpm>	N/A	N/A	N/A	Req	Opt
Average Heart Rate	integer beats per minute	<bpm>140</bpm>	Opt	Opt	Opt	Opt	Opt

Element	Format	Example	Tread-mill	Ellip-tical	Stepper	Bike	Other
		</workoutSummary>	Req	Req	Req	Req	Req
iPod Info							
		<iPodInfo>	Written by iPod				
File Open Date/Time	date/time format w/ offset	<openTime> 2007-12-17T 17:15:17-08:00 </openTime>	Written by iPod				
File Close Date/Time	date/time format w/ offset	<closeTime> 2007-12-17T 17:56:07-08:00 </closeTime>	Written by iPod				
iPod Model	string	<model>MA980 </model>	Written by iPod				
iPod Software Version	string	<softwareVersion> 1.1.2 </softwareVersion>	Written by iPod				
iPod Serial No.	string	<serialNumber> 4H802998TVS </serialNumber>	Written by iPod				
		</iPodInfo>	Written by iPod				
XML Signature							
iPod signature		<Signature >same as run data</Signature>	Written by iPod				
Root Close							
		</gymData>	Written by iPod				

**Note:** To obtain a <manufacturerID> number, a partner must submit a product plan to Apple's Made For iPod program. Once the product plan is approved, Apple will assign the partner an ID for all the partner's Nike + iPod products.

## User Interface Requirements

This section sets forth the requirements for the cardio equipment's user interface. An example of a typical user interface process, from the perspective of the cardio equipment, is shown in [Figure E-1](#) (page 509).

## The iPod Does Not Support Nike + iPod

---

The cardio equipment must determine whether the attached iPod supports the Nike + iPod feature; see [“Determining iPod Support for Cardio Equipment”](#) (page 495). If the attached iPod does not support the Nike + iPod feature, then the cardio equipment must display this information to the user. See [Table E-6](#) (page 508) for the approved UI string.

## Deciding to Record a Workout to the iPod

---

The cardio equipment must first check the iPod for the user’s Workout Recording Preference, using the Sports lingo. If the user’s recording preference is set to Never then the workout must not be recorded. If the user’s preference is set to Always, Ask, or Not Set, then the cardio equipment must present an option to the user. There are two approved forms for this option:

- Recommended: allow user to opt out. Default to recording the workout, but present an option for the user to cancel the recording.
- Allowed: present a dialog prompting the user to choose whether or not they would like the workout to be recorded.

## The iPod is Full

---

In the event that the cardio equipment is unable to record a workout to the iPod because of insufficient disk space, the cardio equipment must present a message to the user alerting them to this problem. See [Table E-6](#) (page 508) for the approved UI string. Note that this problem may occur either when trying to open a workout file or mid-workout, if the iPod’s file system becomes full.

## User Weight

---

Workouts are displayed at `nikeplus.com` using a metric known as Cardio Miles. Cardio Miles are a conversion from workout calories to equivalent running miles. The key metric recorded in most cardio equipment workouts is the calories burned. To accurately calculate calories burned, the user must be required to enter a weight value.

Because a user’s weight may change over time, the equipment must always present the user with an opportunity to enter or adjust the weight value. The enter-weight UI on the cardio equipment must start with the weight value retrieved from iPod via the `GetUserData` command. It is acceptable for the cardio equipment UI to automatically accept the displayed weight if the user fails to enter or adjust the weight after a length of time has passed.

If the user enters a weight different from that retrieved from the iPod, the cardio equipment must write the new weight value back to iPod via the `SetUserData` command.

## Recording Indicator

---

Cardio equipment must always present a visible indicator to the user that the workout is being successfully recorded to iPod. This indicator must be present as soon as the workout recording begins and then disappear when the workout recording ends (either because the workout is finished or the iPod has been detached from the equipment).

Whenever the user pauses a workout, the recording indicator must flash on and off until the workout either resumes normally or is terminated.

## Workout Completed

---

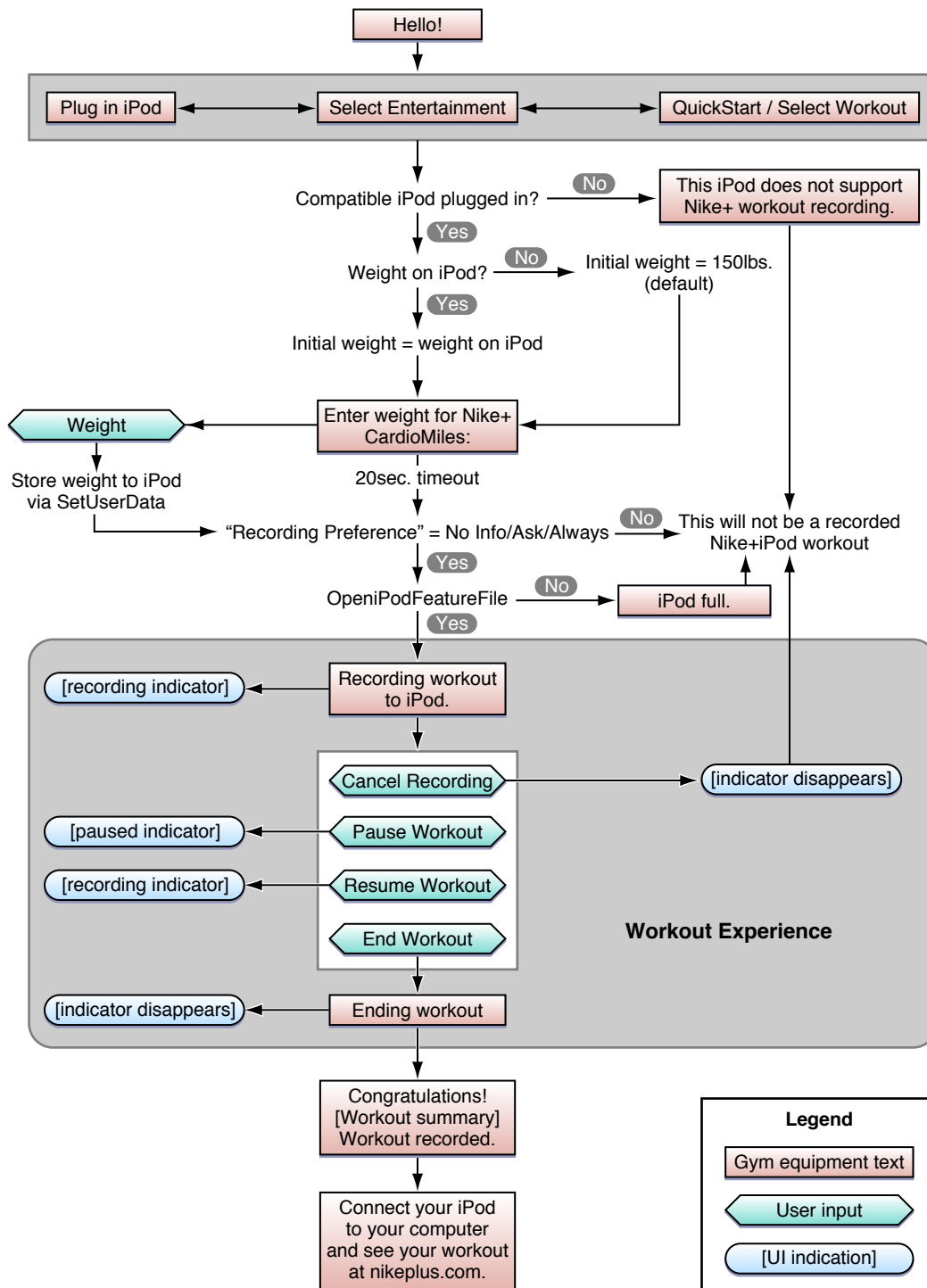
At the end of a workout, the cardio equipment must present two messages to the user. The first is a congratulatory message stating that the workout has been successfully recorded and showing a summary of the workout data. The second message gives brief instruction on how to view the user's workout statistics at the [nikeplus.com](http://nikeplus.com) website. [Table E-6](#) (page 508) lists all the approved UI strings for display by Nike + iPod cardio equipment.

**Table E-6**     Approved user interface messages

Event	Approved UI string	Short UI string
iPod Does Not Support the Nike + iPod System	This iPod does not support Nike+ workout recording.	This iPod does not support workout recording.
The iPod is full	This iPod is full.	iPod full.
Workout summary	Congratulations! <Workout summary> Workout recorded.	Congratulations! <Workout summary> Workout recorded.
Upload instructions	Connect your iPod to your computer to see your workout at <a href="http://nikeplus.com">nikeplus.com</a> .	See your workout at <a href="http://nikeplus.com">nikeplus.com</a> .



Figure E-1 Sample user experience flow chart



## Sample Command Sequence

Table E-7 (page 510) shows a sample sequence of IDPS commands that implements the Nike + iPod cardio equipment system in an accessory.

**Table E-7** Cardio equipment command sequence using IDPS

Step	Accessory command	iPod command	Comment
1	StartIDPS		no params
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::StartIDPS'
<p>If the ACK reply to StartIDPS returns a status of 0x00, the accessory proceeds to Step 3. Otherwise,</p> <ul style="list-style-type: none"> <li>■ If the accessory receives no ACK command within 1 second, it may retry Step 1 at 1-second intervals as long as iPod Detect (pin 30 of the 30-pin connector) is low and Accessory Power (pin 13) is high. Accessories that don't monitor these signals (such as Bluetooth accessories or USB accessories without a 191 kΩ ID resistor) may retry without checking them.</li> <li>■ If the accessory receives an ACK status of 0x04 (Bad Parameter), the iPod does not support IDPS. Within 800 ms the accessory must send an IdentifyDeviceLingoes command with its lingo mask set for only the General lingo, with no options, and with a Device ID of 0x0. See <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83). A sample command sequence is listed in Table F-28 (page 548).</li> <li>■ If the accessory receives any other nonzero ACK status, it must not send any more iAP commands until it has been disconnected. Upon reconnection it must restart identification at Step 1.</li> </ul>			
3	GetiPodOptions-ForLingo		getting options for General Lingo
4		RetiPodOptions-ForLingo	returning options of 000000000003F3FF (Line out usage   Video output   NTSC video signal format   PAL video signal format   Composite video out connection   S-Video video out connection   Component video out connection   Closed captioning (video)   Video aspect ratio 4:3 (fullscreen)   Video aspect ratio 6:9 (widescreen)   reserved   app communication capable   iPod notifications) for General Lingo
5	GetiPodOptions-ForLingo		getting options for Sports Lingo
6		RetiPodOptions-ForLingo	returning options of 0000000000000002 (Nike + iPod cardio equipment) for Sports Lingo

Step	Accessory command	iPod command	Comment
7	GetiPodOptions-ForLingo		getting options for Storage Lingo
8		RetiPodOptions-ForLingo	returning options of 0000000000000003 (iTunes tagging   Nike + iPod cardio equipment) for Storage Lingo
9	SetFIDTokenValues		setting 12 FID tokens ; IdentifyToken = (lingoes: 0/9/12   options 0x00000002   device ID 0x00000200); AccCapsToken = 0x0000000000000805; AccInfoToken = Acc name (FitBicycle 2010); AccInfoToken = Acc FW version (v1.0.1); AccInfoToken = Acc HW version (v1.0.0); AccInfoToken = Acc manufacturer (Apple); AccInfoToken = Acc model number (MA1234LL/A); iPodPreferenceToken = (setting 'used' for preference class 'line out usage' with restore on exit 'selected'); iPodPreferenceToken = (setting 'on' for preference class 'video out setting' with restore on exit 'selected'); iPodPreferenceToken = (setting 'fullscreen' for preference class 'screen configuration' with restore on exit 'selected'); iPodPreferenceToken = (setting 'NTSC' for preference class 'video format setting' with restore on exit 'selected'); iPodPreferenceToken = (setting 'component' for preference class 'video connection' with restore on exit 'selected')
10		RetFIDTokenValueACKs	12 ACKs for FID tokens ; IdentifyToken = accepted; AccCapsToken = accepted; AccInfoToken = (Acc name) accepted; AccInfoToken = (Acc FW version) accepted; AccInfoToken = (Acc HW version) accepted; AccInfoToken = (Acc manufacturer) accepted; AccInfoToken = (Acc model number) accepted; iPodPreferenceToken = (line out usage) accepted; iPodPreferenceToken = (video out setting) accepted; iPodPreferenceToken = (screen configuration) accepted; iPodPreferenceToken = (video format setting) accepted; iPodPreferenceToken = (video connection) accepted

Step	Accessory command	iPod command	Comment
11	EndIDPS		status 'finished with IDPS; proceed to authentication'
12		IDPSStatus	status 'ready for auth'
13		GetDevAuthentication-Info	no params
14	RetDevAuthentication-Info		returning auth protocol v2.0; current section index: 0; maximum section index: 1; cert data ...
15		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
16	RetDevAuthentication-Info		returning auth protocol v2.0; current section index: 1; maximum section index: 1; cert data ...
17		AckDevAuthentication-Info	acknowledging 'auth info supported'
18		GetDeviceCaps	no params
19		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
20	RetDevAuthentication-Signature		returning signature '...'
21		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
22		GetDeviceCaps	requesting accessory Sports lingo capabilities; no params
23		GetDeviceCaps	requesting accessory Storage lingo capabilities; no params
24	RetDeviceCaps		returning 'Gym equipment command support'
25	RetDeviceCaps		returning 'file system type 'none'; lingo v1.02'
26	Get iPodCaps		requesting iPod Sports lingo capabilities; no params
27		Ret iPodCaps	returning 'Gym equipment support   User data support' with userCount of 1

Step	Accessory command	iPod command	Comment
28	GetiPodCaps		requesting iPod Storage lingo capabilities; no params
29		RetiPodCaps	returning 'total space 2147483648; max file size 1073741824; max write size 500; v1.02'
30	GetUserData		requesting 'Gender'
31		RetUserData	returning 'No information' for data type 'Gender'
32	GetUserData		requesting 'Age'
33		RetUserData	returning '26 years' for data type 'Age'
34	GetUserData		requesting 'Name'
35		RetUserData	returning 'NewUser' for data type 'Name'
36	GetUserData		requesting 'Weight'
37		RetUserData	returning '92.0 kg' for data type 'Weight'
38	OpeniPodFeatureFile		opening feature type 'Gym Equipment Workout' with options mask 'file data   ipodInfo   XML signature' and file data:</gymData>;
39		iPodAck	acknowledging 'Success' to command 'Storage Lingo::OpeniPodFeatureFile' with file handle 0
40	WriteiPodFileData		writing 39 bytes at offset 0 and handle 0: <?xml version=""1.0"" encoding=""UTF-8""?>
41		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
42	WriteiPodFileData		writing 10 bytes at offset 39 and handle 0: <gymData>;
43		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
44	WriteiPodFileData		writing 15 bytes at offset 49 and handle 0: <vers>1</vers>;

Step	Accessory command	iPod command	Comment
45		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
46	WriteiPodFileData		writing 166 bytes at offset 64 and handle 0: <equipmentInfo> < manufacturerID>123A3A27 </manufacturerID> < manufacturerName>Scotty </manufacturerName> < type>Bike</type> < model>Bike for Scotty</model> </equipmentInfo>
47		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
48	WriteiPodFileData		writing 36 bytes at offset 230 and handle 0: <userInfo> < kg>90</kg> </userInfo>
49		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
50	WriteiPodFileData		writing 74 bytes at offset 266 and handle 0: <template> < templateName>Goal </templateName> < kCal>500</kCal> </template>
51		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
52	WriteiPodFileData		writing 121 bytes at offset 340 and handle 0: <interval> < event>start</event> < sec>0</sec> < kCal>0</kCal> < km>0</km> < rpm>0</rpm> < bpm>0</bpm> < level>1</level> </interval>
53		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
54	WriteiPodFileData		writing 88 bytes at offset 461 and handle 0: <interval> < sec>10</sec> < kCal>0</kCal> < km>2</km> < rpm>74</rpm> < bpm>96</bpm> </interval>

Step	Accessory command	iPod command	Comment
55		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
56	WriteiPodFileData		writing 90 bytes at offset 549 and handle 0: <interval> < sec>20</sec> < kCal>2</kCal> < km>10</km> < rpm>98</rpm> < bpm>163</bpm> </interval>
57		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
58	WriteiPodFileData		writing 126 bytes at offset 639 and handle 0: <interval> < event>end</event> < sec>30</sec> < kCal>500</kCal> < km>17</km> < rpm>87</rpm> < bpm>172</bpm> < level>2</level> </interval>
59		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
60	WriteiPodFileData		writing 104 bytes at offset 765 and handle 0: <workoutSummary> < sec>30</sec> < kCal>500</kCal> < km>17</km> < rpm>89</rpm> < bpm>167</bpm> </workoutSummary>
61		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
62	CloseiPodFile		closing file with handle 0
63		iPodAck	acknowledging 'Success' to command 'Storage Lingo::CloseiPodFile' with file handle 0





# Historical Information

This appendix memorializes the specifications of past iPod models and iAP protocols. It is included in this specification to provide guidance for developers who need to design accessories compatible with these past technologies.

## Past Protocol Features

[Table F-1](#) (page 517) and [Table F-2](#) (page 519) list the past protocol versions for each lingo and the firmware releases in which they were available.

In the following tables, "NL" indicates that the given lingo was not implemented in the specified model loaded with the specified firmware. "NV" indicates that although the lingo was implemented, its protocol version could not be read from the iPod. Footnotes are listed below each table.

**Table F-1** Past iPod models, firmware, and lingo versions, table 1

Model	Firmware		Lingoes										
	Version	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x0A	0x0C
3G <sup>1</sup>	2.0.0	04/03	NV	NL	NV	NL	NL	NL	NL	NL	NL	NL	NL
	2.1.0	10/03	NV	NV	NV	NL	1.00	NL	NL	NL	NL	NL	NL
	2.2.0	02/04	NV	NV	NV	NL	1.02	NL	NL	NL	NL	NL	NL
mini	1.0.0	02/04	NV	NL	NV	NL	1.01	NL	NL	NL	NL	NL	NL
	1.1.0	03/04	NV	NL	NV	NL	1.03	NL	NL	NL	NL	NL	NL
	1.2.0	11/04	1.00	NL	1.00	NL	1.05	1.00	NL	NL	NL	NL	NL
	1.3.0	02/05	1.00	NL	1.00	NL	1.05	1.00	NL	NL	NL	NL	NL
	1.4.0	06/05	1.02	NL	1.00	1.01	1.05	1.00	NL	NL	NL	NL	NL
4G	3.0.0	07/04	NV	NV	NV	NL	1.04	NV	NL	NL	NL	NL	NL
	3.0.1	08/04	NV	NV	NV	NL	1.04	NV	NL	NL	NL	NL	NL
	3.0.2	11/04	1.00	1.00	1.00	NL	1.05	1.00	NL	NL	NL	NL	NL
	3.1.0	06/05	1.02	1.00	1.00	1.01	1.05	1.00	NL	NL	NL	NL	NL

Model	Firmware		Lingoes										
	Version	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x0A	0x0C
4G (color display)	1.0.0	10/04	1.00	1.00	1.00	NL	1.05	1.00	NL	NL	NL	NL	NL
	1.1.0	03/05	1.01	1.00	1.00	1.00	1.06	1.00	NL	NL	NL	NL	NL
	1.2.0	06/05	1.02	1.00	1.00	1.01	1.06	1.00	NL	NL	NL	NL	NL
nano	1.0.0	09/05	1.02	NL	1.00	1.02	1.07	1.00	NL	1.00	1.00	NL	NL
	1.1.0	01/06	1.03	NL	1.00	1.03	1.09	1.00	NL	1.00	1.00	NL	NL
	1.1.1	03/06	1.03	NL	1.00	1.03	1.09	1.00	NL	1.00	1.00	NL	NL
	1.2.0	06/06	1.04	NL	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 <sup>2</sup>	NL
	1.3	10/06	1.05	NL	1.02	1.05	1.11	1.01	NL	1.00	1.00	1.01	NL
5G	1.0.0	10/05	1.03	1.01	1.01	1.03	1.08	1.00	1.00	1.00	1.00	NL	NL
	1.1.0	01/06	1.03	1.01	1.01	1.03	1.09	1.00	1.00	1.00	1.00	NL	NL
	1.1.1	03/06	1.03	1.01	1.01	1.03	1.09	1.00	1.00	1.00	1.00	NL	NL
	1.1.2	06/06	1.03	1.01	1.01	1.03	1.09	1.00	1.00	1.00	1.00	NL	NL
	1.2.0	09/06	1.05	1.01	1.02	1.05	1.11	1.01	1.00	1.00	1.00	1.01	NL
	1.2.1	11/06	1.05	1.01	1.02	1.05	1.11	1.01	1.00	1.00	1.00	1.01	NL
	1.2.3	11/07	1.06	1.01	1.02	1.05	1.12	1.01	1.00	1.00	1.00	1.01	1.01
2G nano	1.0.0	09/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 <sup>2</sup>	NL
	1.0.1	09/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 <sup>2</sup>	NL
	1.0.2	09/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 <sup>2</sup>	NL
	1.1	10/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 <sup>2</sup>	NL
	1.1.1	10/06	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.00 <sup>2</sup>	NL
	1.1.2	02/07	1.04	1.01	1.02	1.04	1.10	1.01	NL	1.00	1.00	1.02	NL
classic	1.0	09/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.1	09/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.2	10/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.3	11/07	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.01
	1.1.1	02/08	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.01

Model	Firmware		Lingoes										
	Version	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x0A	0x0C
	1.1.2	05/08	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.02
3G nano	1.0	09/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.1	09/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.2	10/07	1.07	1.01	1.02	1.05	1.12	1.01	NL	1.00	1.00	1.03	1.01
	1.0.3	11/07	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.01
	1.1	01/08	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.01
	1.1.2	05/08	1.07	1.01	1.02	1.05	1.13	1.01	NL	1.00	1.00	1.03	1.02

<sup>1</sup> Supporting the 3G iPod requires special design considerations. See “Interfacing With the 3G iPod” in *iPod/iPhone Hardware Specifications*.

<sup>2</sup> Because version 1.00 of Lingo 0x0A contained a bug that was corrected in version 1.01, accessories should attempt Digital Audio only with iPods whose Lingo 0x0A version is higher than or equal to 1.01. See [Table 3-218](#) (page 294).

**Table F-2** Past iPod models, firmware, and lingo versions, table 2

Model	Firmware		Lingoes											
	Vers.	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x07	0x08	0x09	0x0A	0x0C	0x0E
iPhone	1.1	09/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.1	09/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.2	11/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.3	01/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.4	02/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.5	07/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.0	07/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.0.1	08/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.1.0	09/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	2.2.1	01/09	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
1G touch	1.1	09/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL

Model	Firmware		Lingoes											
	Vers.	Date	0x00	0x01	0x02	0x03	0x04	0x05	0x07	0x08	0x09	0x0A	0x0C	0x0E
	1.1.1	09/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.2	11/07	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.3	01/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.4	02/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	1.1.5	07/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.0	07/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.0.1	08/08	1.07	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	NL	NL
	2.1.0	09/08	1.08	NL	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
iPhone 3G	2.1.0	09/08	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	2.2.1	01/09	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.00	NL	1.02	1.02	NL
	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
4G nano	1.0.2	09/08	1.08	1.02	1.03	1.05	1.14	1.01	1.01	1.00	1.01	1.03	1.02	NL
	1.0.3	01/09	1.08	1.02	1.04	1.05	1.14	1.01	1.01	1.00	1.01	1.03	1.02	NL
120 GB classic	2.0.0	09/08	1.08	1.02	1.02	1.05	1.13	1.01	1.00	1.00	NL	1.03	1.02	NL
	2.0.1	01/09	1.08	1.02	1.03	1.05	1.13	1.01	1.00	1.00	NL	1.03	1.02	NL
2G touch	2.1.1	09/08	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	NL
	2.2.1	01/09	1.08	1.02	1.02	1.05	1.12	1.01	NL	1.01	NL	1.02	1.02	NL
	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00
iPhone 3GS	3.0.0	06/09	1.09	1.02	1.02	1.05	1.12	1.01	NL	1.00	1.01	1.02	1.02	1.00

## 9-Pin Audio/Remote Connector Commands

The commands documented in this section apply only to the 9-pin Audio/Remote connector. For the top connector microphone only, the iPod sends a `BeginRecord` command when recording is about to begin. The device microphone bias, if applicable, should already be present. iPod sends an `EndRecord` command when recording is completed. The device may remove microphone bias, if applicable, after the `EndRecord` command is received.

The iPod sends a `BeginPlayback` command when playback is about to begin. The device may then turn on its speaker amplifier, if present. Upon receipt of an `EndPlayback` command, the device must turn off its speaker amplifier. For all Microphone lingo commands sent by the iPod, no device response is expected.

## Command 0x00: BeginRecord

Direction: iPod to Device

The iPod sends this command to notify the device that audio recording has started. The device does not return a packet to the iPod in response to this command. See ["Command 0x06: iPodModeChange"](#) (page 151) for more details.

**Table F-3** `BeginRecord` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x00	Command ID: <code>BeginRecord</code>
5	0xFD	Checksum

## Command 0x01: EndRecord

Direction: iPod to Device

The iPod sends this command to notify the device that audio recording has ended. The device does not return a packet to the iPod in response to this command. See ["Command 0x06: iPodModeChange"](#) (page 151) for more details.

**Table F-4** `EndRecord` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x01	Command ID: <code>EndRecord</code>
5	0xFC	Checksum

## Command 0x02: BeginPlayback

---

Direction: iPod to Device

The iPod sends this command to notify the device that audio playback has started. The device does not return a packet to the iPod in response to this command. See "[Command 0x06: iPodModeChange](#)" (page 151) for more details.

**Table F-5** BeginPlayback packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x02	Command ID: BeginPlayback
5	0xFB	Checksum

## Command 0x03: EndPlayback

---

Direction: iPod to Device

The iPod sends this command to notify the device that audio playback has ended. The device does not return a packet to the iPod in response to this command. See "[Command 0x06: iPodModeChange](#)" (page 151) for more details.

**Table F-6** EndPlayback packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet payload
3	0x01	Lingo ID: Microphone lingo
4	0x03	Command ID: EndPlayback
5	0xFA	Checksum

## General Lingo Command 0x01: Identify (Deprecated)

Direction: Device to iPod

**This command is deprecated.** It should be used only by accessories that need to support the 3G iPod. Supporting the 3G iPod requires other special design considerations; see “Interfacing With the 3G iPod” in *iPod/iPhone Hardware Specifications*.

Accessories that use this command send it to notify the iPod that an accessory has been attached and to register the lingo it supports. Accessories should identify at boot time and any time they receive “[Command 0x00: RequestIdentify](#)” (page 64) from the iPod. The iPod does not send an `ACK` command in response.

**Note:** The `Identify` command should be used only over the UART serial port link.

Accessories should follow the identification guidelines in “[Packet Signaling and Initialization Using the UART Serial Port Link](#)” (page 48) and “[iAP Signaling and Initialization Using the USB Port Link](#)” (page 50) to guarantee they have established communication with the iPod when using this command. Accessory devices that support more than one lingo (not including the General lingo) or that plan to use the USB transport should use the IDPS process.

The `Identify` command disables all but free lingoes on the current port. For serial ports, this means lingoes 0x00, 0x02, 0x03, and 0x05 may be used, excluding authenticated commands; the USB port will be able to use only the general lingo, 0x00 (see [Table 2-4](#) (page 54)). Devices that register with this command can use only the General lingo (0x00) commands plus those of the specific lingo that they identified, with a few exceptions. All devices may use the Simple Remote lingo (0x02) `ContextButtonStatus` command (0x00). If the identified lingo is the Extended Interface lingo (0x04), the device may also use the Display Remote lingo (0x03) commands if that lingo is not already being used by another device.

If the lingo identified by the `Identify` command can be used by only one device at a time, and that lingo is already in use by a different device, the command will fail but no command failure `ACK` command will be sent to the device. The device can verify that the `Identify` command has succeeded by sending a free command with valid parameters and checking the iPod’s response. The iPod will respond with an `ACK` response with failure status if the lingo is already in use.

This command performs lingo conflict checking to ensure that single-instance lingoes, such as Display Remote, are used on only one port at a time.

**Table F-7**      `Identify` packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet payload
3	0x00	Lingo ID: General lingo
4	0x01	Command: <code>Identify</code>

Byte number	Value	Comment
5	0x0N	Supported lingo. For example, if the device supports the Simple Remote lingo, this byte should be 0x02.
6	0xNN	Checksum

The `Identify` command has facilities for devices to draw more than 5 mA power from the iPod. The `Identify` command sent by such a device contains the supported lingo and the optional power bitfields described in [Table F-9](#) (page 524). These bits define the power requirements of the device. [Table F-8](#) (page 524) shows the format of an `Identify` command for a high-power device.

**Table F-8** `Identify` packet for high-power devices

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x06	Packet payload length
3	0x00	Lingo ID: General lingo
4	0x01	Command: <code>Identify</code>
5	0x05	Supported lingo: Accessory Power lingo
6	0x00	Reserved: set to 0x00
7	0x02	Number of valid bits in the power option flag
8	0x0N	Option flag bits. See <a href="#">Table F-9</a> (page 524).
9	0xNN	Checksum

The option flag byte consists of bitfields. At this time, the most significant 6 bits of the option flags are reserved and should be zero. Bit 1 is defined for Apple use only and must be zero. Bit 0 should be 1 if the device requires more than 5 mA at any time. This may be the case if it is powered by the iPod.

**Table F-9** Power option bits

Bit	Description
0	Power consumption requirements. Possible values are: 0 = device requires 5 mA or less power from iPod at all times 1 = device requires more than 5 mA power from iPod (up to 100mA maximum) during playback operation
1	Reserved. This bit must be set to 0 by external devices.
2–7	Reserved. Set to 0.



## Lingo 0x06: USB Host Control Lingo

**Note:** This lingo is deprecated; do not use it in new products.

When an accessory is attached to an iPod, the iPod ordinarily acts as the USB device and the accessory acts as the USB host. The USB Host Control lingo lets the accessory switch roles with the iPod, becoming the USB device to the iPod's host. The USB Host Control lingo also automatically launches the Photo Import application on the iPod.

When an accessory identifies itself as using this lingo, the roles are switched automatically. If authentication fails or if the accessory is detached from the iPod, the iPod reverts to being the USB device.

**Note:** This lingo may be used only over the serial port link, not over the USB port link.

The accessory developer may choose to build a self-powered accessory that communicates with the iPod over USB or may choose to build a USB dongle similar to the Apple iPod Camera Connector.

[Table F-10](#) (page 525) summarizes the USB Host Control commands.

**Table F-10** USB Host Control lingo command summary

Command	ID	Direction	Data length	Protocol version	Authentication required
ACK (command/status)	0x00	Dev to iPod	4	1.00	Yes
GetUSBPowerState	0x01	iPod to Dev	2	1.00	Yes
RetUSBPowerState	0x02	Dev to iPod	3	1.00	Yes
SetUSBPowerState	0x03	iPod to Dev	3	1.00	Yes
Reserved	0x04–0xFF	N/A	N/A	N/A	N/A

Selected Host Control lingo command timeout information and the number of times the command tries to execute are shown in [Table F-11](#) (page 525).

**Table F-11** Select Host Control command timings

Lingo	Command	Timeout	Tries
USB Host Control (0x06)	GetUsbPowerState (0x01)	500 ms	1
	SetUsbPowerState (0x03)	500 ms	1

## Command History of the USB Host Control Lingo

[Table F-12](#) (page 526) shows the history of changes to the USB Host Control lingo.

**Table F-12** USB Host Control lingo command history

Lingo version	Command changes	Features
1.00	Add: 0x00–0x03	USB power state support

## Command 0x00: ACK

---

Direction: Device to iPod

This command is sent by the device in response to a command received from the iPod. It reports the original command number and the status of the command.

**Table F-13** ACK packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x04	Length of packet
3	0x06	Lingo ID: USB Host Control lingo
4	0x00	Command ID: ACK
5	0xNN	Command status (see <a href="#">Table 3-28</a> (page 169))
6	0xNN	ID of the command being acknowledged
7	0xNN	Checksum

## Command 0x01: GetUSBPowerState

---

Direction: iPod to Device

This command is sent by the iPod to obtain the device's current USB power state. In response, the device must send a `RetUSBPowerState` command with the current USB power setting.

**Table F-14** GetUSBPowerState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x02	Length of packet
3	0x06	Lingo ID: USB Host Control lingo

Byte number	Value	Comment
4	0x01	Command ID: GetUSBPowerState
5	0xF7	Checksum

## Command 0x02: RetUSBPowerState

---

Direction: Device to iPod

This command is sent by the device in response to a `GetUSBPowerState` command received from the iPod. It returns the current state of the USB power supply.

**Table F-15** RetUSBPowerState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x06	Lingo ID: USB Host Control lingo
4	0x02	Command ID: RetUSBPowerState
5	0xNN	Current power state (zero for off, nonzero for on)
6	0xNN	Checksum

## Command 0x03: SetUSBPowerState

---

Direction: iPod to Device

This command is sent by the iPod to set the device's USB power state. The device must set the USB power state and respond with an `ACK` command indicating command completion.

**Table F-16** SetUSBPowerState packet

Byte number	Value	Comment
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet (SOP)
2	0x03	Length of packet
3	0x06	Lingo ID: USB Host Control lingo
4	0x03	Command ID: SetUSBPowerState

Byte number	Value	Comment
5	0xNN	Power state to be set (zero for off, nonzero for on)
6	0xNN	Checksum

## Deprecated Extended Interface Commands

The commands in this section have been replaced by commands in the iAP General lingo (Lingo 0x00).

### Command 0x0012: RequestProtocolVersion

Direction: Device to iPod

**Note:** This command is deprecated except for use by accessories supporting 3G iPods. Use the General lingo (0x00) command 0x0F, `RequestLingoProtocolVersion`, instead.

Requests the version of the running protocol from the iPod. The iPod responds with a "[Command 0x0013: ReturnProtocolVersion](#)" (page 528) command.

**Note:** This command requests the Extended Interface protocol version, not the iPod software version.

**Table F-17** `RequestProtocolVersion` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x12	Command ID (bits 7:0)
6	0xE7	Packet payload checksum byte

### Command 0x0013: ReturnProtocolVersion

Direction: iPod to Device

**Note:** This command is deprecated except for use by accessories supporting 3G iPods. Use the General lingo (0x00) command 0x10, `ReturnLingoProtocolVersion`, instead.

Returns the iPod Extended Interface protocol version number. The iPod sends this command in response to the "[Command 0x0012: RequestProtocolVersion](#)" (page 528) command from the device. The major version number specifies the protocol version digits to the left of the decimal point; the minor version number specifies the digits to the right of the decimal point. For example, a major version number of 0x01 and a minor version number of 0x08 represents an Extended Interface protocol version of 1.08. This protocol information is also available through the General lingo (lingo 0x00) command `RequestLingoProtocolVersion` when passing lingo 0x04 as the lingo parameter.

**Note:** This command returns the Extended Interface protocol version, not the iPod software version.

**Table F-18** `ReturnProtocolVersion` command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x05	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x13	Command ID (bits 7:0)
6	0xNN	Protocol major version number
7	0xNN	Protocol minor version number
8	0xNN	Packet payload checksum byte

## Command 0x0014: RequestiPodName

Direction: Device to iPod

**Note:** This command is deprecated except for use by accessories supporting 3G iPods. Use the General lingo (0x00) command 0x07, `RequestiPodName`, instead.

Returns the name of the user's iPod or "iPod" if the iPod name is undefined. This allows the iPod name to be shown in the human-machine interface (HMI) of the interfacing body. The iPod responds with the "[Command 0x0015: ReturniPodName](#)" (page 530) command containing the iPod name text string.

**Table F-19** RequestiPodName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0x03	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x14	Command ID (bits 7:0)
6	0xE5	Packet payload checksum byte

## Command 0x0015: ReturniPodName

Direction: iPod to Device

**Note:** This command is deprecated except for use by accessories supporting 3G iPods. Use the General lingo (0x00) command 0x08, `ReturniPodName`, instead.

The iPod sends this command in response to the "[Command 0x0014: RequestiPodName](#)" (page 529) command from the device. The iPod name is encoded as a null-terminated UTF-8 character array. The iPod name string is not limited to 252 characters; it may be sent in small or large packet format. The small packet format is shown.

**Note:** Starting with version 1.07 of the Extended Interface lingo, the `ReturniPodName` command on Windows-formatted iPods returns the iTunes name of the iPod instead of the Windows volume name.

**Table F-20** ReturniPodName command

Byte number	Value	Meaning
0	0xFF	Sync byte (required only for UART serial)
1	0x55	Start of packet
2	0xNN	Packet payload length
3	0x04	Lingo ID: Extended Interface lingo
4	0x00	Command ID (bits 15:8)
5	0x15	Command ID (bits 7:0)
6...N	0xNN	iPod name as UTF-8 character array

Byte number	Value	Meaning
(last byte)	0xNN	Packet payload checksum byte

## Authentication 1.0 Sample Command Sequence

For legacy purposes, [Table F-21](#) (page 531) shows the process implemented by some existing devices, using authentication 1.0. This process is not supported for new device designs.

**Table F-21** Legacy accessory authentication

Step	Action or command	Direction	Comments
1	IdentifyDeviceLingoes (0x13)	Device to iPod	The accessory identifies itself, lists its supported lingoes, and requests authentication. It can request immediate authentication or it can defer authentication until it tries to use restricted lingoes or commands.
2	ACK (0x02)	iPod to Device	The iPod acknowledges receipt of IdentifyDeviceLingoes.
5	GetDevAuthentication-Info (0x14)	iPod to Device	The iPod requests accessory authentication information and starts its timeout timer.
6	RetDevAuthentication-Info (0x15)	Device to iPod	The accessory returns its major and minor authentication version.
7	AckDevAuthentication-Info (0x16)	iPod to Device	The iPod returns the status of its check of the authentication version. If it does not support the authentication version, the iPod sends a value of 0x08 to the accessory.
The iPod lingoes and commands are enabled after the authentication version is validated.			
8	GetAccessoryInfo (0x27)	iPod to Device	The iPod queries the accessory for information about it.
9	RetAccessoryInfo (0x28)	Device to iPod	The accessory returns information about its identity and capabilities.
10	GetDevAuthentication-Signature (0x17)	iPod to Device	The iPod sends a 16-byte random challenge to the accessory and asks it to calculate a digital signature.
11	RetDevAuthentication-Signature (0x18)	Device to iPod	The accessory returns its digital signature to the iPod within 7.5 seconds.
12	AckDevAuthentication-Status (0x19)	iPod to Device	The iPod verifies the signature by deriving a public key from the Device ID and returns the status of signature verification.

## Accessory Identification With Non-IDPS iPods

Some models of iPods and iPhones do not support IDPS. Every accessory must always start its identification process by sending `StartIDPS`; if the iPod's ACK response passes a `status` value of 0x04 (Bad Parameter), the accessory may revert to an identification process using `IdentifyDeviceLingoes`.

**Note:** The accessory must send `IdentifyDeviceLingoes` within 800 ms of receiving the iPod's ACK response.

The sample command listings in this section illustrate various forms of the identification process using `IdentifyDeviceLingoes`:

- [Table F-22](#) (page 532) shows the basic process for accessories that communicate using the UART port link.
- [Table F-23](#) (page 534) shows the basic process for accessories that communicate using USB or Bluetooth.
- [Table F-24](#) (page 535) lists sample commands for an accessory that supports the General and Extended Interface lingoes. For Extended Interface command details, see "[The Extended Interface Protocol](#)" (page 341).
- [Table F-25](#) (page 539) lists sample commands for an accessory that supports the General, Simple Remote, and Display Remote lingoes.
- [Table F-26](#) (page 542) lists sample commands for an accessory that supports the General, Simple Remote, Display Remote, and Digital Audio lingoes.
- [Table F-27](#) (page 545) shows a sample sequence of commands that implements radio tagging in an accessory. See "[iTunes Tagging](#)" (page 463).
- [Table F-28](#) (page 548) shows a sample sequence of commands that implements the Nike + iPod cardio equipment system in an accessory. See "[Nike + iPod Cardio Equipment System](#)" (page 495).

**Table F-22**    UART accessory identification with non-IDPS iPods

Step	Action or command	Direction	Comments
1	Wait 80 ms after the iPod turns on Accessory Power (pin 13 in "Hardware Interfaces" in <i>iPod/iPhone Hardware Specifications</i> )	Device	Wait for the iPod's internal bootstrap and wakeup. If the iPod is in Sleep mode, the accessory must repeat Steps 1-5 until the iPod wakes and the accessory receives an ACK command.
2	Send sync byte (0xFF)	Device to iPod	Allow iPod to synchronize to the device's baud rate.
3	Wait 20 ms	Device	
4	<code>StartIDPS</code> (0x38)	Device to iPod	The accessory sends <code>StartIDPS</code> to start the identification process specified in " <a href="#">Accessory Identification</a> " (page 445).



Step	Action or command	Direction	Comments
5	Wait up to 1 second	Device	The device waits for the iPod to send an ACK command acknowledging receipt of StartIDPS.
6	ACK (0x02) of StartIDPS	iPod to Device	The iPod sends a status of 0x04 (Bad Parameter) to indicate that it does not support IDPS; see <a href="#">Table 2-1</a> (page 48), Step 6.
7	IdentifyDevice-Lingoes (0x13)	Device to iPod	The accessory sends IdentifyDevice-Lingoes with the lingo mask set to only the General lingo, the option bit mask set to 0x0, and the Device ID set to 0x0. This cancels any active authentication process, as detailed in <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83).
8	Wait up to 1 second	Device	The device waits for the iPod to send an ACK command acknowledging receipt of IdentifyDeviceLingoes.
9	ACK (0x02) of IdentifyDevice-Lingoes	iPod to Device	The iPod sends a status of 0x00 (OK) to acknowledge receipt of the empty IdentifyDeviceLingoes command.
<p>If the iPod does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an Identify command to achieve limited use of iAP. See <a href="#">General Lingo Command 0x01: Identify (Deprecated)</a> (page 523) and "Interfacing With the 3G iPod" in <i>iPod/iPhone Hardware Specifications</i>.</p>			
10	GetAccessoryInfo (0x27)	iPod to Device	The iPod queries the accessory for information about it.
11	RetAccessoryInfo (0x28)	Device to iPod	The accessory returns an empty string. Because the device immediately sends a second IdentifyDeviceLingoes command (Step 9), the iPod may acknowledge the RetAccessoryInfo command with an ACK command indicating an error status. The device must ignore this acknowledgment.
12	IdentifyDevice-Lingoes (0x13)	Device to iPod	The accessory sends a second IdentifyDeviceLingoes command, specifying which lingoes it supports and requesting immediate authentication (see <a href="#">"Authentication"</a> (page 52)).

Step	Action or command	Direction	Comments
13	ACK (0x02) of IdentifyDeviceLingoes	iPod to Device	The iPod acknowledges receipt of the second IdentifyDeviceLingoes command.
The iPod or iPhone now initiates the authentication process by sending a GetDevAuthenticationInfo command to the accessory, as shown in <a href="#">Table 2-5</a> (page 55).			

**Table F-23** USB or BT accessory identification with non-IDPS iPods

Step	Action or command	Direction	Comments
1	StartIDPS (0x38)	Device to iPod	The accessory sends StartIDPS to start the identification process specified in <a href="#">"Accessory Identification"</a> (page 445).
2	Wait up to 1 second	Device	The device waits for the iPod to send an ACK command acknowledging receipt of StartIDPS.
3	ACK (0x02) of StartIDPS	iPod to Device	The iPod sends a status of 0x04 (Bad Parameter) to indicate that it does not support IDPS; see <a href="#">Table 2-2</a> (page 51), Step 3.
4	IdentifyDeviceLingoes (0x13)	Device to iPod	The accessory sends IdentifyDeviceLingoes with the lingo mask set to only the General lingo, the option bit mask set to 0x0, and the Device ID set to 0x0. This cancels any active authentication process, as detailed in <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83).
5	Wait up to 1 second	Device	The device waits for the iPod to send an ACK command acknowledging receipt of IdentifyDeviceLingoes.
6	ACK (0x02) of IdentifyDeviceLingoes	iPod to Device	The iPod sends a status of 0x00 (OK) to acknowledge receipt of the empty IdentifyDeviceLingoes command.
7	GetAccessoryInfo (0x27)	iPod to Device	The iPod queries the accessory for information about it.
8	RetAccessoryInfo (0x28)	Device to iPod	The accessory returns an empty string. Because the device immediately sends a second IdentifyDeviceLingoes command (Step 9), the iPod may acknowledge the RetAccessoryInfo command with an ACK command indicating an error status. The device must ignore this acknowledgment.

Step	Action or command	Direction	Comments
9	IdentifyDevice-Lingoes (0x13)	Device to iPod	The accessory sends a second IdentifyDevice-Lingoes command, specifying which lingoes it supports and requesting immediate authentication (see <a href="#">"Authentication"</a> (page 52)).
10	ACK (0x02) of IdentifyDevice-Lingoes	iPod to Device	The iPod acknowledges receipt of the second IdentifyDeviceLingoes command.
The iPod or iPhone now initiates the authentication process by sending a GetDevAuthenticationInfo command to the accessory, as shown in <a href="#">Table 2-5</a> (page 55).			

**Table F-24** Non-IDPS identification of lingoes 0x00+0x04

Step	Accessory command	iPod command	Comment
Steps A–D cancel any current authentication process; see <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83). Step B also detects if the attached iPod is a 3G iPod.			
A	IdentifyDevice-Lingoes		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the iPod to respond.			
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
If the iPod does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an Identify command to achieve limited use of iAP. See <a href="#">General Lingo Command 0x01: Identify (Deprecated)</a> (page 523) and "Interfacing With the 3G iPod" in <i>iPod/iPhone Hardware Specifications</i> .			
C		GetAccessoryInfo	no params
D	RetAccessoryInfo		returning an empty string
The accessory is now assured that it can successfully identify its lingoes to the attached iPod or iPhone.			
1	IdentifyDevice-Lingoes		identifying lingoes 'General Extended Interface'; options 'auth:immediate; power:low; device ID 0x00000200
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
3		GetDevAuthentication-Info	no params

Step	Accessory command	iPod command	Comment
4	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1;
5		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
6	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1;
7		AckDevAuthentication-Info	acknowledging 'auth info supported'
8		GetAccessoryInfo	requesting 'Acc info capabilities'
9		GetDevAuthentication-Signature	offering challenge
10	RetAccessoryInfo		returning 'Acc info capabilities Acc name Acc FW version Acc HW version Acc manufacturer Acc model number Acc serial number Acc incoming max packet size' in response to 'Acc info capabilities'
11		GetAccessoryInfo	requesting 'Acc name'
12	RetDevAuthentication-Signature		returning signature
13	RetAccessoryInfo		returning 'Apple' in response to 'Acc name'
14		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
15		GetAccessoryInfo	requesting 'Acc FW version'
16	RetAccessoryInfo		returning 'v9.8.7' in response to 'Acc FW version'
17		GetAccessoryInfo	requesting 'Acc HW version'
18	RetAccessoryInfo		returning 'v1.2.3' in response to 'Acc HW version'
19		GetAccessoryInfo	requesting 'Acc manufacturer'
20	RetAccessoryInfo		returning 'Apple Inc.' in response to 'Acc manufacturer'
21		GetAccessoryInfo	requesting 'Acc model number'

Step	Accessory command	iPod command	Comment
22	RetAccessoryInfo		returning 'ModelNumber-XYZ' in response to 'Acc model number'
23		GetAccessoryInfo	requesting 'Acc serial number'
24	RetAccessoryInfo		returning 'SerialNumber-1234' in response to 'Acc serial number'
25		GetAccessoryInfo	requesting 'Acc incoming max packet size'
26	RetAccessoryInfo		returning '1024 bytes' in response to 'Acc incoming max packet size'
27	EnterRemoteUIMode		no params
28		ACK	acknowledging 'Command Pending (max wait time: 3000 ms)' to command 'General Lingo::EnterRemoteUIMode'
29		ACK	acknowledging 'Success (OK)' to command 'General Lingo::EnterRemoteUIMode'
30	RequestRemoteUIMode		no params
31		ReturnRemoteUIMode	returning 'Extended Interface Mode'
32	ResetDBSelection		no params
33		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::ResetDBSelection'
34	GetNumberCategorized-DBRecords		for category 'Playlist'
35		ReturnNumber-CategorizedDBRecords	returning 'record count 22'
36	RetrieveCategorized-DatabaseRecords		requesting 'category Playlist record start index 0 record read count 4'
37		ReturnCategorized-DatabaseRecord	returning 'record category index 0 MyiPhone'
38		ReturnCategorized-DatabaseRecord	returning 'record category index 1 Purchased'
39		ReturnCategorized-DatabaseRecord	returning 'record category index 2 Against Doctor's Orders'

Step	Accessory command	iPod command	Comment
40		ReturnCategorized-DatabaseRecord	returning 'record category index 3 Always In Your Mind
41	SelectDBRecord		selecting 'type Playlist record index 3
42		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::SelectDBRecord'
43	PlayCurrentSelection		playing selection track index 0
44		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::PlayCurrentSelection'
45	SetPlayStatusChange-Notification		setting 'basic play state changes track index'
46		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::SetPlayStatusChangeNotification'
47	PlayControl		sending 'Toggle Play/Pause'
48		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::PlayControl'
49	PlayControl		sending 'Next Track'
50		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::PlayControl'
51		PlayStatusChange-Notification	notifying new play status 'playback track changed' (new track record index 1)
52	PlayControl		sending 'Toggle Play/Pause'
53		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::PlayControl'
54	SetCurrentPlaying-Track		setting currently playing track to 3
55		ACK	acknowledging 'Success (OK)' to command 'Extended Interface Lingo::SetCurrentPlayingTrack'
56		PlayStatusChange-Notification	notifying new play status 'playback track changed' (new track record index 3)

Step	Accessory command	iPod command	Comment
57	GetPlayStatus		no params
58		ReturnPlayStatus	returning 'Playing track length 155506 ms track position 15637 ms'
59	GetIndexedPlaying-TrackTitle		requesting title name for track index 3
60		ReturnIndexedPlaying-TrackTitle	returning Auld Lang Syne
61	GetIndexedPlaying-TrackArtistName		requesting artist name for track index 3
62		ReturnIndexedPlaying-TrackArtistName	returning Straight No Chaser
63	GetIndexedPlaying-TrackAlbumName		requesting album name for track index 3
64		ReturnIndexedPlaying-TrackAlbumName	returning Holiday Spirits (Bonus Track Version)
65	ExitRemoteUIMode		no params
66		ACK	acknowledging 'Command Pending (max wait time: 3000 ms)' to command 'General Lingo::ExitRemoteUIMode'
67		ACK	acknowledging 'Success (OK)' to command 'General Lingo::ExitRemoteUIMode'

**Table F-25** Non-IDPS identification of lingo 0x00+0x02+0x03

Step	Accessory command	iPod command	Comment
Steps A–D cancel any current authentication process; see <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83). Step B also detects if the attached iPod is a 3G iPod.			
A	IdentifyDevice-Lingoes		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the iPod to respond.			
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'

Step	Accessory command	iPod command	Comment
If the iPod does not send an <b>ACK</b> command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an <b>ACK</b> command after the third try, it may either quit or send an <b>Identify</b> command to achieve limited use of iAP. See <a href="#">General Lingo Command 0x01: Identify (Deprecated)</a> (page 523) and “Interfacing With the 3G iPod” in <i>iPod/iPhone Hardware Specifications</i> .			
C		GetAccessoryInfo	no params
D	RetAccessoryInfo		returning an empty string
The accessory is now assured that it can successfully identify its lingo to the attached iPod or iPhone.			
1	IdentifyDevice-Lingo		identifying lingo 'General Simple Remote Display Remote'; options 'auth:immediate; power:low; device ID 0x00000200'
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingo'
3		GetDevAuthentication-Info	no params
4	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1;
5		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
6	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1;
7		AckDevAuthentication-Info	acknowledging 'auth info supported'
8		GetAccessoryInfo	requesting 'Acc info capabilities'
9		GetDevAuthentication-Signature	offering challenge
10	RetAccessoryInfo		returning 'Acc info capabilities Acc name Acc FW version Acc HW version Acc manufacturer Acc model number Acc serial number Acc incoming max packet size' in response to 'Acc info capabilities'
11		GetAccessoryInfo	requesting 'Acc name'



Step	Accessory command	iPod command	Comment
12	RetDevAuthentication-Signature		returning signature
13	RetAccessoryInfo		returning 'Apple' in response to 'Acc name'
14		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
15		GetAccessoryInfo	requesting 'Acc FW version'
16	RetAccessoryInfo		returning 'v9.8.7' in response to 'Acc FW version'
17		GetAccessoryInfo	requesting 'Acc HW version'
18	RetAccessoryInfo		returning 'v1.2.3' in response to 'Acc HW version'
19		GetAccessoryInfo	requesting 'Acc manufacturer'
20	RetAccessoryInfo		returning 'Apple Inc.' in response to 'Acc manufacturer'
21		GetAccessoryInfo	requesting 'Acc model number'
22	RetAccessoryInfo		returning 'ModelNumber-XYZ' in response to 'Acc model number'
23		GetAccessoryInfo	requesting 'Acc serial number'
24	RetAccessoryInfo		returning 'SerialNumber-1234' in response to 'Acc serial number'
25		GetAccessoryInfo	requesting 'Acc incoming max packet size'
26	RetAccessoryInfo		returning '1024 bytes' in response to 'Acc incoming max packet size'
27	SetRemoteEvent-Notification		setting 'Track playback index Play status'
28		ACK	acknowledging 'Success (OK)' to command 'Display Remote Lingo::SetRemoteEventNotification'
29	GetIndexedPlaying-TrackInfo		getting info type 'Track title' for track index 0 and chapter index 0
30		RetIndexedPlaying-TrackInfo	returning 'Always In Your Mind' for info type 'Track title'

Step	Accessory command	iPod command	Comment
31	GetPlayStatus		no params
32		RetPlayStatus	returning 'Playback paused index 3 length 155506 ms position 84933 ms'
33	ContextButtonStatus		sending status 'Play/Pause'
34	ContextButtonStatus		sending status 'All buttons up'
35		RemoteEvent-Notification	reporting status 'Playing' for parameter 'Play status'
36	GetPlayStatus		no params
37		RetPlayStatus	returning 'Playing index 3 length 155506 ms position 91644 ms'
38	ContextButtonStatus		sending status 'Next Track'
39	ContextButtonStatus		sending status 'All buttons up'
40		RemoteEvent-Notification	reporting status 'index 4' for parameter 'Track playback index'
41	SetCurrentPlaying-Track		setting index 0
42		ACK	acknowledging 'Success (OK)' to command 'Display Remote Lingo::SetCurrentPlayingTrack'
43		RemoteEvent-Notification	reporting status 'index 0' for parameter 'Track playback index'
44	GetIndexedPlaying-TrackInfo		getting info type 'Track title' for track index 0 and chapter index 0
45		RetIndexedPlaying-TrackInfo	returning 'Always In Your Mind' for info type 'Track title'

**Table F-26** Non-IDPS identification of lingo 0x00+0x02+0x03+0x0A

Step	Accessory command	iPod command	Comment
Steps A–D cancel any current authentication process; see <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingo"</a> (page 83). Step B also detects if the attached iPod is a 3G iPod.			
A	IdentifyDevice-Lingo		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the iPod to respond.			

Step	Accessory command	iPod command	Comment
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
<p>If the iPod does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an Identify command to achieve limited use of iAP. See <a href="#">General Lingo Command 0x01: Identify (Deprecated)</a> (page 523) and “Interfacing With the 3G iPod” in <i>iPod/iPhone Hardware Specifications</i>.</p>			
C		GetAccessoryInfo	no params
D	RetAccessoryInfo		returning an empty string
The accessory is now assured that it can successfully identify its lingoes to the attached iPod or iPhone.			
1	IdentifyDevice-Lingoes		identifying lingoes 'General  Simple Remote  Display Remote  Digital Audio'; options 'auth:immediate; power:low; device ID 0x00000200'
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
3		GetDevAuthentication-Info	no params
4	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1;
5		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
6	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1;
7		AckDevAuthentication-Info	acknowledging 'auth info supported'
8		GetAccSampleRateCaps	no params
9		GetAccessoryInfo	requesting 'Acc info capabilities'
10		GetDevAuthentication-Signature	offering challenge
11	RetAccSampleRateCaps		returning sample rates '8000  11025  12000  16000  22050  24000  32000  44100  48000'

Step	Accessory command	iPod command	Comment
12	RetAccessoryInfo		returning 'Acc info capabilities  Acc name  Acc FW version  Acc HW version  Acc manufacturer  Acc model number  Acc serial number  Acc incoming max packet size' in response to 'Acc info capabilities'
13	RetDevAuthentication-Signature		returning signature
14		GetAccessoryInfo	requesting 'Acc name'
15	RetAccessoryInfo		returning 'Apple' in response to 'Acc name'
16		NewiPodTrackInfo	sample rate 44100; Sound Check value 0; track volume adjustment 0
17	AccAck		acknowledging 'Success (OK)' to command 'Digital Audio Lingo::NewiPodTrackInfo'
18		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
19		GetAccessoryInfo	requesting 'Acc FW version'
20	RetAccessoryInfo		returning 'v9.8.7' in response to 'Acc FW version'
21		GetAccessoryInfo	requesting 'Acc HW version'
22	RetAccessoryInfo		returning 'v1.2.3' in response to 'Acc HW version'
23		GetAccessoryInfo	requesting 'Acc manufacturer'
24	RetAccessoryInfo		returning 'Apple Inc.' in response to 'Acc manufacturer'
25		GetAccessoryInfo	requesting 'Acc model number'
26	RetAccessoryInfo		returning 'ModelNumber-XYZ' in response to 'Acc model number'
27		GetAccessoryInfo	requesting 'Acc serial number'
28	RetAccessoryInfo		returning 'SerialNumber-1234' in response to 'Acc serial number'
29		GetAccessoryInfo	requesting 'Acc incoming max packet size'

Step	Accessory command	iPod command	Comment
30	RetAccessoryInfo		returning '1024 bytes' in response to 'Acc incoming max packet size'

**Table F-27** Non-IDPS radio tagging command sequence

Step	Accessory command	iPod command	Comment
Steps A–D cancel any current authentication process; see <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83). Step B also detects if the attached iPod is a 3G iPod.			
A	IdentifyDeviceLingoes		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the iPod to respond.			
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
If the iPod does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an Identify command to achieve limited use of iAP. See <a href="#">General Lingo Command 0x01: Identify (Deprecated)</a> (page 523) and "Interfacing With the 3G iPod" in <i>iPod/iPhone Hardware Specifications</i> .			
C		GetAccessoryInfo	no params
D	RetAccessoryInfo		returning an empty string
The accessory is now assured that it can successfully identify its lingoes to the attached iPod or iPhone.			
1	IdentifyDeviceLingoes		identifying lingoes 'General'; options 'auth:none; power:low'; device ID 0x00000000
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
3	GetiPodOptions		no params
4		RetiPodOptions	returning 'video output line out'
5	RequestLingoProtocolVersion		requesting General Lingo version
6		ReturnLingoProtocolVersion	returning General Lingo v1.09
7	RequestLingoProtocolVersion		requesting Storage Lingo version

Step	Accessory command	iPod command	Comment
8		ReturnLingoProtocol-Version	returning Storage Lingo v1.02
9	IdentifyDevice-Lingoes		identifying lingoes 'General Storage'; options 'auth:immediate; power:low'; device ID 0x00000200
10		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
11		GetDevAuthentication-Info	no params
12	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1; cert data: ...
13		ACK	acknowledging 'Success (OK)' to command 'General Lingo::RetDevAuthenticationInfo'
14	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1; cert data: ...
15		AckDevAuthentication-Info	acknowledging 'auth info supported'
16		GetAccessoryInfo	requesting 'Acc info capabilities'
17		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
18	RetDevAuthentication-Signature		returning signature '...'
19		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
20	RetAccessoryInfo		returning 'Acc info capabilities Acc name Acc FW version Acc HW version Acc manufacturer Acc model number Acc incoming max packet size' in response to 'Acc info capabilities'
21		GetAccessoryInfo	requesting 'Acc name'
22	RetAccessoryInfo		returning 'Radio' in response to 'Acc name'
23		GetAccessoryInfo	requesting 'Acc FW version'

Step	Accessory command	iPod command	Comment
24	RetAccessoryInfo		returning 'v1.0.0' in response to 'Acc FW version'
25		GetAccessoryInfo	requesting 'Acc HW version'
26	RetAccessoryInfo		returning 'v1.0.0' in response to 'Acc HW version'
27		GetAccessoryInfo	requesting 'Acc manufacturer'
28	RetAccessoryInfo		returning 'Radio Manufacturer' in response to 'Acc manufacturer'
29		GetAccessoryInfo	requesting 'Acc model number'
30	RetAccessoryInfo		returning 'M78901LL/Z' in response to 'Acc model number'
31		GetAccessoryInfo	requesting 'Acc incoming max packet size'
32	RetAccessoryInfo		returning '2048 bytes' in response to 'Acc incoming max packet size'
33	GetiPodCaps		no params
34		RetiPodCaps	returning 'total space 2147483648; max file size 1073741824; max write size 500; v1.02'
35	GetiPodFreeSpace		no params
36		RetiPodFreeSpace	returning 130023424 bytes
37	OpeniPodFeatureFile		opening feature type 'Radio Tagging'
38		RetiPodFileHandle	returning file handle 0
39		iPodAck	acknowledging 'Success' to command 'Storage Lingo::WriteiPodFileData' with file handle 0
40	CloseiPodFile		closing file with handle 0
41		iPodAck	acknowledging 'Success' to command 'Storage Lingo::CloseiPodFile' with file handle 0

**Table F-28** Non-IDPS cardio equipment command sequence

Step	Accessory command	iPod command	Comment
Steps A–D cancel any current authentication process; see <a href="#">"Cancelling a Current Authentication Process With IdentifyDeviceLingoes"</a> (page 83). Step B also detects if the attached iPod is a 3G iPod.			
A	IdentifyDevice-Lingoes		identifying lingo 'General'; options none; device ID 0x00000000
The accessory waits up to 1 second for the iPod to respond.			
B		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
If the iPod does not send an ACK command within 1 second, it may be a 3G iPod. In this case, if the accessory needs to support the 3G iPod and it is using UART communication, it may retry Step A up to 3 times, for a total elapsed time of 3 seconds. If the accessory does not receive an ACK command after the third try, it may either quit or send an Identify command to achieve limited use of iAP. See <a href="#">General Lingo Command 0x01: Identify (Deprecated)</a> (page 523) and "Interfacing With the 3G iPod" in <i>iPod/iPhone Hardware Specifications</i> .			
C		GetAccessoryInfo	no params
D	RetAccessoryInfo		returning an empty string
The accessory is now assured that it can successfully identify its lingoes to the attached iPod or iPhone.			
1	IdentifyDevice-Lingoes		identifying lingoes 'General'; options 'auth:none; power:low'; device ID 0x00000000
2		ACK	acknowledging 'Success (OK)' to command 'General Lingo::IdentifyDeviceLingoes'
3	GetiPodOptions		no params
4		RetiPodOptions	returning 'video output line out'
5	RequestLingoProtocol-Version		requesting General Lingo version
6		ReturnLingoProtocol-Version	returning General Lingo v1.09
7	RequestLingoProtocol-Version		requesting Sports Lingo version
8		ReturnLingoProtocol-Version	returning Sports Lingo v1.01
9	RequestLingoProtocol-Version		requesting Storage Lingo version



Step	Accessory command	iPod command	Comment
10		ReturnLingoProtocol-Version	returning Storage Lingo v1.02
11	IdentifyDevice-Lingoes		identifying lingoes 'General Sports Storage'; options 'auth:immediate; power:low'; device ID 0x00000200
12		ACK	acknowledging 'Success (OK)' to command 'General Lingo:: IdentifyDeviceLingoes'
13		GetDevAuthentication-Info	no params
14	RetDevAuthentication-Info		returning auth protocol v2.0; section: 0/1; cert data: ...
15		ACK	acknowledging 'Success (OK)' to command 'General Lingo:: RetDevAuthenticationInfo'
16	RetDevAuthentication-Info		returning auth protocol v2.0; section: 1/1; cert data: ...
17		AckDevAuthentication-Info	acknowledging 'auth info supported'
18		GetDeviceCaps	no params
19		GetAccessoryInfo	requesting 'Acc info capabilities'
20		GetDevAuthentication-Signature	offering challenge '...' with retry counter 1
21	RetDevAuthentication-Signature		returning signature '...'
22		AckDevAuthentication-Status	acknowledging authentication status 'Success (OK)'
23		GetDeviceCaps	no params
24	RetDeviceCaps		returning 'Gym equipment command support' with max 0 node filters
25		GetDeviceCaps	no params
26	RetDeviceCaps		returning 'total space 0; max file size 0; max write size 0; read-only; no subdirs; random writes allowed; max file count 0; max name length 0; file system type Reserved; v1.02'

Step	Accessory command	iPod command	Comment
27	RetAccessoryInfo		returning 'Acc info capabilities  Acc name  Acc FW version  Acc HW version  Acc manufacturer  Acc model number  Acc incoming max packet size' in response to 'Acc info capabilities'
28		GetAccessoryInfo	requesting 'Acc name'
29	RetAccessoryInfo		returning 'Bike' in response to 'Acc name'
30		GetAccessoryInfo	requesting 'Acc FW version'
31	RetAccessoryInfo		returning 'v1.0.0' in response to 'Acc FW version'
32		GetAccessoryInfo	requesting 'Acc HW version'
33	RetAccessoryInfo		returning 'v1.0.0' in response to 'Acc HW version'
34		GetAccessoryInfo	requesting 'Acc manufacturer'
35	RetAccessoryInfo		returning 'Bike Manufacturer' in response to 'Acc manufacturer'
36		GetAccessoryInfo	requesting 'Acc model number'
37	RetAccessoryInfo		returning 'M78901LL/Z' in response to 'Acc model number'
38		GetAccessoryInfo	requesting 'Acc incoming max packet size'
39	RetAccessoryInfo		returning '2048 bytes' in response to 'Acc incoming max packet size'
40	Get iPodCaps		no params
41		Ret iPodCaps	returning 'Gym equipment support  User data support' with userCount of 1
42	Get iPodCaps		no params
43		Ret iPodCaps	returning 'total space 2147483648; max file size 1073741824; max write size 500; read-only; no subdirs; random writes allowed; max file count 10000; max name length 200; file system type HFS+; v1.02'
44	GetUserIndex		no params

Step	Accessory command	iPod command	Comment
45		RetUserIndex	returning userIndex of 0
46	GetUserData		requesting 'Preferred unit system'
47		RetUserData	returning 'No information' for data type 'Preferred unit system'
48	GetUserData		requesting 'Name'
49		RetUserData	returning 'NewUser' for data type 'Name'
50	GetUserData		requesting 'Gender'
51		RetUserData	returning 'No information' for data type 'Gender'
52	GetUserData		requesting 'Weight'
53		RetUserData	returning '92.0 kg' for data type 'Weight'
54	GetUserData		requesting 'Age'
55		RetUserData	returning '26 years' for data type 'Age'
56	GetUserData		requesting 'Recording preference'
57		RetUserData	returning 'No information' for data type 'Recording preference'
58	OpeniPodFeatureFile		opening feature type 'Gym Equipment Workout' with options mask 'file data  ipodInfo  XML signature' and file data:</gymData>;
59		RetiPodFileHandle	returning file handle 0
60	WriteiPodFileData		writing 293 bytes at offset 0 and handle 0: <?xml version=""1.0"" encoding=""UTF-8""?>; <gymData>; <vers>1</vers>; <equipmentInfo>; <manufacturerID>00000001</manufacturerID>; <manufacturerName>Bike Manufacturer</manufacturerName>; <type>Bike</type>; <model>M78901LL/Z</model>; <serialNumber> UV1234567890-SN</serialNumber>; </equipmentInfo>;

Step	Accessory command	iPod command	Comment
61		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
62	WriteiPodFileData		writing 125 bytes at offset 240 and handle 0: <userInfo><kg>92.0</kg></userInfo>; <template>; <templateName>Athletic Challenge</templateName>; <sec>60</sec>; </template>;
63		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
64	WriteiPodFileData		writing 125 bytes at offset 366 and handle 0: <interval>; <event>start</event>; <sec>0</sec>; <kCal>0</kCal>; <km>0.00</km>; <rpm>26</rpm>; <level>1</level>; </interval>;
65		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
66	WriteiPodFileData		writing 86 bytes at offset 392 and handle 0: <interval>; <sec>10</sec>; <kCal>0</kCal>; <km>0.02</km>; <rpm>64</rpm>; </interval>;
67		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
68	WriteiPodFileData		writing 87 bytes at offset 479 and handle 0: <interval>; <sec>20</sec>; <kCal>12</kCal>; <km>0.7</km>; <rpm>189</rpm>; </interval>;
69		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
70	WriteiPodFileData		writing 107 bytes at offset 567 and handle 0: <interval>; <event>end</event>; <sec>21</sec>; <kCal>13</kCal>; <km>0.8</km>; <rpm>170</rpm>; </interval>;

Step	Accessory command	iPod command	Comment
71		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
72	WriteiPodFileData		writing 99 bytes at offset 675 and handle 0: <workoutSummary>; <sec>21</sec>; <kCal>13</kCal>; <km>0.8</km>; <rpm>170</rpm>; </workoutSummary>;
73		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: WriteiPodFileData' with file handle 0
74	CloseiPodFile		closing file with handle 0
75		iPodAck	acknowledging 'Success' to command 'Storage Lingo:: CloseiPodFile' with file handle 0



# Glossary

---

**accessory** A third-party device licensed under the Made for iPod or Works With iPhone program. See **device**.

**authentication** A mechanism used by an iPod to verify whether an attached device is an authorized accessory and by an accessory to authenticate the iPod, if desired.

**checksum** The byte sum of packet bytes from the payload length through the last packet byte. This is used to validate the contents of a command packet. For a valid packet, the sum of the bytes, including the checksum byte, must be 0x00. The packet checksum byte—the last byte in a packet—must be the 2's complement (the negative) of the sum of the payload length byte up to, but not including, the packet checksum byte.

**deprecated** Used to describe a technology or feature that is supported but whose use is discouraged and not recommended. Such a technology or feature has typically been replaced by a newer one and is likely to become unsupported in the future.

**device** An external electronic component connected to the iPod using the 30-pin connector or Bluetooth.

**FID** A Full ID string, used by IDPS to send token-value fields that the iPod is able to parse during the accessory identification process.

**HID (Human Interface Device)** HID is a standard USB class. A USB host such as a PC or Macintosh will recognize any attached USB device that supports a HID interface and makes it available to the application layers of the operating system via a set of programming interfaces. A common application of a HID interface is a USB mouse or joystick.

**HID report** A single unit of data that is used to send information to the HID interface of the iPod or from the iPod to the host. iAP packets are broken into HID reports before being sent across the USB port link and are reassembled on the receiving side.

**IDPS** Identify Device Preferences and Settings, the identification process required for an accessory to communicate with an iPod or iPhone. See "[Accessory Identification](#)" (page 445).

**iUI (iPod USB Interface)** A configuration of the iPod when attached as a device over USB. This configuration allows the iPod to be controlled using iAP, using a USB HID class interface as a transport mechanism.

**LCB (Link Control Byte)** A byte used by the iUI to indicate report sets and manage data flow.

**lingo** The command category used by a device. There is a General lingo that must be supported by all devices. Other lingo's are designed for use by specific devices, such as simple remote controls and microphones.

**link** The logical connection between an external device and the iPod via serial port or other physical connection.

**packet** The logical set of bytes that compose a valid command sequence. This set includes the packet start byte, packet payload length, payload, and payload checksum. Note that a sync byte is appended to the beginning of the packet when using the UART serial port as the data transport link. There are two different packet types: small format and large format.

**payload** The sequence of bytes consisting of the lingo, command, and data that are contained within a packet.

**podcasting** A way to publish multimedia files on the Internet that lets users receive new files automatically by subscription. Podcast files are typically downloaded to iPods through Apple's iTunes application.

**RDS/RBDS (Radio [Broadcast] Display System)** A technology for broadcasting and displaying artist, album, track titles, and similar information on FM radio receivers.

**resistor-based accessory** An accessory that uses an Accessory Identify resistor to access only limited functions in an iPod. Compare Serial accessory.

**RSSI (Receive Signal Strength Indicator)** A measure of the strength of an RF signal coming into a radio frequency tuner.

**serial accessory** An accessory that uses the iPod Accessory Protocol Interface to access a range of iPod functions. Compare Resistor-based accessory.

**UART (Universal Asynchronous Receiver/Transmitter)** A piece of computer hardware that translates between parallel and serial bits of data. A UART is usually an integrated circuit used for serial communications over a computer or peripheral device serial port.

**USB (Universal Serial Bus)** An interface standard for communication between a computer and external peripherals over a cable using biserial transmission.

**USB descriptor** A standard USB data structure that is passed from a USB device to the host upon request. Descriptors are used by the USB device to communicate its characteristics and resource requirements to the host.

**USB endpoint** A logical connection point that is used to set up a data transfer pipe between a USB host and interface on a device. For instance, the HID interface on the iPod uses an interrupt-type endpoint to enable a pipe for transferring data to the USB Host.

**USB host** A single computer connected to one or more USB devices or functions. The host is responsible for recognizing that a USB device has been attached to it and for driving the communications with the device. For the purposes of this document, the iPod is a USB device that provides a function, and the accessory is the USB host.

**X.509 certificate** A standard defined by the International Telecommunication Union (ITU) that governs the format of certificates used for authentication and sender identity verification in public-key cryptography. In the iAP, X.509 certificates contain the public keys used in the authentication process.



# Document Revision History

This table describes the changes to *iPod Accessory Protocol Interface Specification*.

Date	Notes
2009-10-22	<i>Revision R38:</i>
	Added section " <a href="#">Reserved Commands and Data</a> " (page 47).
	Added information for the 5G nano, the 2G touch (2009), and the iPod classic 160 GB.
	Added commands 0x49-0x4A, 0x4D-0x4F, and 0x51 to the General lingo to support Event Notifications; see <a href="#">Table 2-9</a> (page 60). Also added explanatory section " <a href="#">iPod Event Notifications</a> " (page 449).
	Added commands 0x0D and 0x0E to the Simple Remote lingo to support button actions for 5G nano Radio Tagging and Camera accessories; see <a href="#">Table 3-17</a> (page 157).
	Added section " <a href="#">Accessory Control of the iPod 5G nano Camera</a> " (page 161).
	Added commands 0x25-0x31 to the RF Tuner lingo, and added features to other commands, to support HD radio; see <a href="#">Table 3-93</a> (page 222).
	Incorporated numerous updates and corrections.
2009-09-09	<i>Revision R37:</i>
	Exported the following chapters and appendixes to new book <i>iPod/iPhone Hardware Specifications</i> , Release R1: "Hardware Interfaces," "Functional Description," "Protocol Transport Links," "iPod Power States and Accessory Power," "Headphone Remote and Mic System," "Interfacing With the 3G iPod," "Sample Accessory Circuits," "Power Guidelines," and "FireWire to USB Reference Design."
	Exported appendix "Headset and FireWire-to-USB Power Converter Certification" to new book <i>iPod/iPhone Accessory Testing and Certification Specification</i> , Release R1.
	Imported entire contents of discontinued book <i>iPod Extended Interface Specification</i> , Release R25, into " <a href="#">The Extended Interface Protocol</a> " (page 341).
	Imported some content from discontinued book <i>iPhone Accessory Interface Specification</i> , Release R9.
	Reformatted content to new layout and typography.

Date	Notes
2009-06-23	<i>Revision R36:</i>
	Added information about the iPhone 3GS.
	Added new appendix " <a href="#">Accessory Identification</a> " (page 445). <i>This identification process is required in all new accessory designs.</i> Moved command examples using <code>IdentifyDeviceLingoes</code> to " <a href="#">Accessory Identification With Non-IDPS iPods</a> " (page 532) in Appendix M, "Historical Information."
	In Chapter 3, added new section " <a href="#">Accessory Communication With iPhone OS Applications</a> " (page 42); deleted section "iPod Games" and updated section " <a href="#">Accessory Control of the iPod touch and iPhone</a> " (page 42).
	Added commands 0x38-0x3C, 0x3F-0x43, and 0x4A-0x4C to the General lingo; see <a href="#">Table 2-9</a> (page 60).
	Added new section " <a href="#">Accessory Power Policy</a> " (page 39).
	Added new section " <a href="#">Lingo 0x0E: Location Lingo</a> " (page 312).
	Added sample command sequences for iTunes tagging ( <a href="#">Table D-11</a> (page 484)) and cardio equipment ( <a href="#">Table E-7</a> (page 510)).
	Added new appendix " <a href="#">Transaction IDs</a> " (page 451).
	Added new appendix " <a href="#">Multisection Data Transfers</a> " (page 459).
	Added new section "Avoiding an iPhone OS Warning When a Self-Powered Accessory Is Off."
	Added caution about latching connectors to "30-Pin Connector."
	Changed button press detection parameters in "iPod/iPhone Headphone/Microphone Jack."
	Added information about PKCS-7 transport of certificate data to " <a href="#">Device Authentication of iPod</a> " (page 57).
	Added warning about the iPod touch and iPhone Off state to "Power States."
	Updated signal level information in "UART Serial Port Link."
	Added new section " <a href="#">Examples of Transaction IDs</a> " (page 456).
	Added new section <a href="#">Sample Identification Sequences Using IdentifyDeviceLingoes</a> (page 334).
2009-03-17	<i>Revision R35:</i>
	Added <a href="#">Table I-1</a> (page 30) to identify iPod/iPhone models.
	Revised "iPhone headphone/microphone schematic" to show iPod/iPhone differences.

Date	Notes
	Revised section "Line Level Input" in Chapter 2.
	Updated <a href="#">Table 1-3</a> (page 37) to show current firmware versions.
	Added display resolution data to <a href="#">Table 1-6</a> (page 41).
	Updated and revised accessory identification and authentication in <a href="#">Table 2-1</a> (page 48) and <a href="#">Table F-21</a> (page 531).
	Added new ACK error codes to <a href="#">Table 2-13</a> (page 65).
	Added copy protection requirement to "USB Audio Transport" (page 291).
	Revised section "Button Detection Circuitry" in Appendix C.
2009-01-05	<i>Revision R34:</i>
	Added definitions of legal agreement terminology (" <a href="#">Specification Terms</a> " (page 34)).
	Added new appendix "Accessory Certification."
	Added FM radio tagging information to " <a href="#">iTunes Tagging</a> " (page 463).
	Added new section " <a href="#">Lingo 0x09: Sports Lingo</a> " (page 277).
	Added new appendix " <a href="#">Nike + iPod Cardio Equipment System</a> " (page 495).
	Added new appendix "Headphone Remote and Mic System."
	Added new commands 0x80-0x82 and new options for command 0x12 in " <a href="#">Lingo 0x0C: Storage Lingo</a> " (page 300).
	Documented new models: the 4G iPod nano, 120 GB classic, and 2G iPod touch.
	Added section " <a href="#">Accessory Communication with the iPod touch and iPhone</a> " (page 42).
	Added hardware details of the iPhone audio connection ("iPod/iPhone Headphone/Microphone Jack").
	Clarified USB power requirements in "USB 2.0."
	Added section "Magnetic Sensitivity of the iPod."
	Defined recommended procedure for determining iPod capabilities in " <a href="#">Command 0x13: IdentifyDeviceLingoes</a> " (page 80).
	Added section " <a href="#">Playback Engine Playlists</a> " (page 159).
	Clarified behavior of Next and Previous buttons in " <a href="#">Using Contextual Buttons</a> " (page 159).
	Clarified mute event in <a href="#">Table 3-51</a> (page 188).

Date	Notes
	Deprecated " <a href="#">Lingo 0x06: USB Host Control Lingo</a> " (page 525).
	Deprecated " <a href="#">General Lingo Command 0x01: Identify</a> " (page 523)).
	Removed from Chapter 5 obsolete description of testing for the General lingo over the UART serial port link.
2008-06-26	<i>Revision R33:</i>
	Added section " <a href="#">Command Timings</a> " (page 146).
	Revised audiobook playback speeds in <a href="#">Table 3-56</a> (page 193).
	Added accessory requirements to <a href="#">Table 2-57</a> (page 98).
	Changed names of levels of authentication from V1 and V2 to authentication 1.0 and 2.0.
	Changed name of RF Transmitter lingo (0x05) to Accessory Power lingo.
	Added section "Supplying USB Power" to Appendix A.
	Changed " <a href="#">Data Transfer to the iPod</a> " (page 479) in Appendix B to show all radio tagging plist fields required.
	Added section "Powering the 3G iPod" to Appendix C.
	Added Note about lyrics strings to <a href="#">Table 3-74</a> (page 207).
	Revised "Typical diode bridge circuit for an AC adapter" in Appendix E.
	Fixed typo in Table F-2 in Appendix F.
2008-05-07	<i>Revision R32:</i>
	Added new appendix, "FireWire to USB Reference Design."
	Revised documentation of sleep states in Chapter 2, Appendix A, and elsewhere.
	Added section " <a href="#">Video Output Preferences</a> " (page 41) and expanded <a href="#">Table 2-64</a> (page 103).
	Updated the current firmware versions in <a href="#">Table 1-2</a> (page 36).
	Made several updates to Appendix B, "iTunes Tagging."
	Made numerous other updates, corrections, and clarifications throughout the document.
2007-12-12	<i>Revision R31:</i>
	Moved past firmware version documentation, description of the 9-pin Audio/Remote connector, FireWire specifications, and other noncurrent material to new appendix, " <a href="#">Historical Information</a> " (page 517).

Date	Notes
	Documented November 2007 firmware for the 5G, classic, 3G nano, and touch iPod models.
	Made minor correction to "Configuration and interface descriptors for iPods with USB audio."
	Simplified document structure.
2007-10-02	<i>Revision R30:</i>
	Added " <a href="#">Lingo 0x0C: Storage Lingo</a> " (page 300) to Chapter 6.
	Added new Appendix B, " <a href="#">iTunes Tagging</a> " (page 463).
2007-09-05	<i>Revision R29:</i>
	Added documentation for the iPod classic, iPod 3G nano, and iPod touch.
	Added documentation for component video outputs.
	Added new command, <code>SetVideoDelay</code> , to the Digital Audio lingo.
	Deprecated the FireWire interface on the 30-pin connector.
	Added new preference IDs to <code>GetiPodPreferences</code> .
	Added design guidelines for third-party developers of AC adapter accessories for the iPod touch ("Power Guidelines").
	Added design guidelines for third-party developers of carrying cases for iPod touch ("iPod touch Carrying Case Design").
2007-06-29	<i>Revision R28:</i>
	Revised pin connections in 30-pin to FireWire cable ("30-pin to FireWire cable").
	Updated model listings to include iPhone and new iPod models ( <a href="#">Table 2-29</a> (page 76))
	Updated requirements for artwork count data ( <a href="#">Table 3-74</a> (page 207))
	Added caution about sending signals to the iPod UART when its serial receive block is off.
	Documented X.509 certificate classes ( <a href="#">Table 2-3</a> (page 53))
	Added line-out usage controls ( <a href="#">Table 2-64</a> (page 103))
	Added section " <a href="#">USB Audio Errors on Older iPods</a> " (page 151)
	Added authentication requirement to General lingo commands 0x1A-0x1F
	Deprecated Level V1 authentication for new designs (see " <a href="#">iPod Authentication of Device</a> " (page 55))

Date	Notes
	Deprecated "Command 0x01: Identify" (page 523)
2007-02-06	<i>Revision R27:</i>
	Added section "Minimizing Crosstalk and Noise."
	Added new information to Table 1-2 (page 36).
	Removed autobaud on parity errors from Table 1-4 (page 38).
	Clarified UART communication rates in "UART Serial Port Link."
	Removed Manufacturer String and Product String from "Choosing an iPod USB Configuration."
	Added example of using iAP over USB in "Transferring IdentifyDeviceLingoes and ACK commands over USB using iAP."
	Distinguished behavior of audio and video playback when iPod enters Extended Interface mode in "Command 0x05: EnterRemoteUIMode" (page 68).
	Added example of using Display Remote Protocol in "Lingo 0x03: Display Remote Lingo" (page 176).
	Clarified usage of NewiPodTrackInfo command in "Command 0x04: NewiPodTrackInfo" (page 298).
	Added new appendix, "Interfacing With the 3G iPod."
	Added new appendix, "Sample Accessory Circuits."
	Added temperature range to "UART Serial Port Link."
2006-11-03	Added new information to Tables 3-1 and 5-26.
2006-10-17	<i>Revision R26:</i>
	Added new information to the note after Table 2-2.
	Updated Table 2-3.
	Added note to section "Line Level Output."
	Added new section "Headphone Jack on Video-Capable iPods."
	Added new iPod models and software versions.
2006-09-12	<i>Revision R25:</i>
	Added iPod options and preferences commands (0x24-0x25 and 0x29-0x2B) to General lingo.
	Added USB Host Control lingo (0x06).

Date	Notes
	Added RF Tuner lingo (0x07).
	Updated list of model ID strings ( <a href="#">Table 2-28</a> (page 75) and <a href="#">Table 2-29</a> (page 76)).
	Corrected RetTrackArtworkData packet listing ( <a href="#">Table 3-81</a> (page 212)).
2006-06-19	<i>Revision R24:</i>
	Added Accessory Equalizer lingo, number 0x08.
	Added USB Digital Audio lingo, number 0x0A.
	Added Authentication level V2 (X.509 certification) to General Lingo (0x00).
	Added Album Art commands (0x16-0x19 and 0x1F-0x20) to Display Remote Lingo (0x03).
	Added GetAccessoryInfo (0x27) and RetAccessoryInfo (0x28) commands to the General lingo (0x00).
	Added Dedicated Media commands (0x00-0x04) to the Simple Remote lingo (0x02).
	Added timeout and retry information to various command descriptions.
	Moved documentation of Extended Interface commands (0x03-0x06, General lingo) from the <i>iPod Extended Interface Specification</i> .
	Revised model and feature tables in Chapter 3.
	Added and updated lingo history tables in Chapter 6.
2006-02-10	<i>Revision R23:</i>
	Pages 17 and 27: Changed audio output power specification to 25 mW.
	Page 87: A simple remote device must send a data payload when all buttons are released; 200 ms timeout removed.
	Corrected <a href="#">Table 1-2</a> (page 36).
	Note, page 17: Specified Technical Note TN001i by name.
2006-01-05	<i>Revision R22:</i>
	General update and reorganization of content.
	Added information on iPod power states.
	Added “D+ and D- connections for a 500 mA USB power brick” and “iPod equivalent input circuits.”
2005-10-12	Updated functional descriptions.

Date	Notes
	Added new microphone lingo commands.
	Additional information about hibernate mode.
	Bug fixes for volume control and others.
2005-09-07	Added support for USB/iUI, authentication and additions to the new Display Remote lingo (0x03)
2005-03-21	Added Equalizer Control lingo support and commands.
2004-11-12	Added general lingo commands, bug fixes and pinouts for the iPod photo release
2004-08-03	Reserved the 28k pulldown resistor
2004-07-21	Added lingo command packet examples.
	Corrected doc properties.
	Added minor clarifications.
	Added new accessory detect image.
	Updated content based on internal review.
2004-05-17	Incorporated review feedback
2004-04-20	Update simple remote, doc reformat
2004-01-14	Minor clarifications
2003-08-12	5mA access power note
2003-08-04	New serial, add bottom serial
2003-07-22	Picture to show Remote Data lines
2003-07-07	License agreement
2003-04-15	Remote protocols added
2003-04-02	Car Charger Detect added
2003-02-04	Accessory Detect Resistor Change
2003-01-09	Rx, Tx Clarification
2002-12-04	Initial Release.